PyXspec

1.1.0

Generated by Doxygen 1.7.5.1

Tue Jul 21 2015 10:41:44

Contents

1 PyXspec Documentation

The source code distribution of XSPEC is required for using PyXspec

1.1 Introduction

PyXspec is an object oriented Python interface to the XSPEC spectral-fitting program. It provides an alternative to Tcl, the sole scripting language for standard Xspec usage. With PyXspec loaded, a user can run Xspec with Python language scripts or interactively at a Python shell prompt.

Not all of the full standard Xspec functionality has been implemented (see What's - Missing). However we will continue to add to this, and we look forward to hearing users' comments and suggestions to help us prioritize the future work.

1.2 About This Manual

The manual contains a Build/Install and Troubleshoot section, a Quick Version tutorial showing basic PyXspec usage, and an Extended Version tutorial for greater functionality. The Quick Version is the recommended starting point for all users.

A class reference guide follows, with descriptions for each of the PyXspec public class methods and attributes. The guide is auto-generated by Doxygen directly from the PyXspec Python code files.

1.3 Authors

PyXspec was developed by Craig Gordon and Keith Arnaud

HEASARC Software Development, Astrophysics Science Division,

Code 660.1, NASA/GSFC, Greenbelt MD 20771

Please send questions, comments, and bug reports to xspec12@athena.gsfc.nasa.gov

2 Release Notes

2.1 Version 1.1.0 Jul 2015 [XSPEC 12.9.0]

2.1.1 New Features

- Local models may now be written in Python and inserted into XSPEC's models library with new **AllModels.addPyMod()** function.
- The plot array retrieval interface (ie. Plot.x(), Plot.y()) has been expanded to allow retrieval from secondary plot panels in a mulit-panel plot.
- New **Parameter.index** attribute.
- New **backscale** attribute for Spectrum and Background classes.
- Added new function **Fit.stepparResults()** for retrieving results of most recent steppar run. (Previously available as a patch)
- New **noWrite** option added to **AllData.fakeit** (Previously available as patch)

2.1.2 Fixes

- The **Model.__call__** function now returns Parameter objects by reference rather than by value. This is to allow the returned object to retain any custom attributes the user may have added.
- Improved handling of **Ctrl-c** breaking in several prompting contexts.

Version 1.0.4 Jul 2014 [XSPEC 12.8.2]

- Added Fit.testStatistic attribute for retrieving the test statistic value from the most recent fit.
- Added compiler macros for switching to <Python/Python.h> include paths when building on Mac platforms.
- Bug fix for get/set Spectrum.correction files on OS X Mavericks.

Version 1.0.3 Aug 2013 [XSPEC 12.8.1]

- The Fit.statMethod and statTest attributes can now be set for a range of individual spectra rather than only applying to all.
- Bug fix to the RModel class (ie. the class of the Response.gain attribute). If the user assigned multiple RModel objects to point to the same underlying XSPEC response gain, changes made through one object weren't necessarily showing up in the other objects. This also fixes a related bug created in Xspec patch 12.8.0l that caused a list of error messages to appear (only) in Python versions 2.6.x.

Version 1.0.2 Jan 2013

- Added Xset.parallel attribute, with options 'leven' and 'error' for setting parallel processes.
- Added Fit.statTest attribute for getting/setting the XSPEC test statistic.

Version 1.0.1 Dec 2012 [XSPEC 12.8.0]

- 2 additions to the **Spectrum** class: an **ignored** attribute and an **ignoredString()** function. The former returns a Python list object containing every ignored channel number. The latter returns the same information in convenient string form, which can be reused as input to a future **ignore** or **notice** command.

2.2 Version 1.0 Feb 2012 [XSPEC 12.7.1]

Changes relative to the PyXspec **Beta** version:

∗∗∗ Important: Two Backwards-Incompatible Changes ∗∗∗

- When using **multiple data groups**, the Model objects assigned to the higher-numbered groups now all have their parameters indexed from **1 to nPar**. For example with a 3 parameter model applied to 2 data groups, you would now access the first parameter in the 2nd model object with "mod2(1)" rather than "mod2(4)".

- The **Model.setPars()** function (introduced with patch 12.7.0f) used the **p*n*** keyword argument syntax to set non-consecutive parameters. This has been replaced. with the use of Python **dictionaries**. For example, m.setPars(p2=.3, p4=1.1) should now be m.setPars({2:.3, 4:1.1}).

2.2.1 New Features

- Added Standard XSPEC's **gain** command functionality. This is implemented with the new **gain** attribute for Response classes. Response.gain is a class of type **RModel**, and has two Parameter objects: **slope** and **intercept**.
- New AllModels.setPars() function for changing multiple parameters in multiple - Model objects with a single call.
- Now compatible with Cygwin.

`Features Previously Added As Patches To XSPEC 12.7.0`

- AllModels.initpackage() for building local models inside the Python shell.
- Bayesian inference provided through the Fit.bayes and Parameter.prior attributes.
- Fit.goodness() and Fit.improve() functions.
- Model.setPars() function for changing multiple parameters with a single call.
- AllModels.simpars() function to do the equivalent of Standard XSPEC's tclout simpars.
- Fit.covariance attribute for retrieving the covariance matrix from the most recent fit.
- Model.expression attribute which stores the model expression string.
- AllModels.sources attribute which stores a map of source number and model name assignments.

2.2.2 Fixes

- All PyXspec bug fixes previously released as patches to XSPEC 12.7.0 are included.
- Now handles model component-by-name access when the component is a table model whose name includes whitespace.

3 Build and Install PyXspec

3.1 Requirements

Since we do not distribute Python with the HEASOFT packages, you'll need to have it already installed on your system (which it is with most Linux and Mac OSX distributions). PyXspec requires a Python version of 2.x where x is 3 or later. The **python** executable must be on your path, and with the library and header files located in the standard directories relative to the executable (see the **Troubleshooting** section for more info).

3.2 Building/Installing

PyXspec is fully integrated into the general HEASOFT build procedure, as described at `http://heasarc.gsfc.nasa.gov/lheasoft/install.html` . So it will be **built and installed automatically** with the rest of XSPEC/HEASOFT, requiring no additional effort from the user.

Once HEASOFT is finished building and installing, you should find PyXspec's code files and lib_pyXspec.so library in the directory $HEADAS/lib/python/xspec.

When you run the HEASOFT initialization script ($HEADAS/headas-init.csh or .sh), it will add $HEADAS/lib/python to your PYTHONPATH environment. This allows Python find the PyXspec module so that you may load it into your session from anywhere, using the "import xspec" statement.

3.3 Running on Mac OS X

These issues apply only to Mac OS X users. Linux users may skip this section.

Beginning with HEASoft-6.16, Mac builds are 64-bit by default. Therefore if you have a default build, you should no longer run Python in 32-bit mode.

Case 1: Running the default **Xcode** distribution of Python (normally /usr/bin/python).

You may use the default Xcode Python if your HEASOFT distribution was built using the **Xcode gcc and g++ compilers**, with only the Fortran compiler coming from a 3rd party such as Fink or MacPorts. But if you built HEASOFT with ALL of your compilers coming from Fink or MacPorts, you cannot use the Xcode Python (see **Case 3**).

HEASoft builds on Macs are now **64-bit by default**. However if (and only if) you forced your build into **32-bit** mode by configuring with --enable-mac_32bit_build=yes, you will need to set:

```
export VERSIONER_PYTHON_PREFER_32_BIT=yes     # Bourne-like shells
    or
setenv VERSIONER_PYTHON_PREFER_32_BIT yes     # C-like shells
```

BEFORE importing the xspec module into Python.

Case 2: Running a Python distribution obtained from **www.python.org**.

If (and only if) you forced your HEASoft build into 32-bit mode, you'll need to ensure that you run the 32-bit Python version and the method used in **Case 1** will not work. Instead, invoke 32-bit Python by way of the **arch** command. For example:

```
arch -i386 python2.7
```

Case 3: Running the **Fink** or **MacPorts** Python.

This applies only to users who have built HEASOFT using Fink or MacPorts for **ALL 3 of their compilers** (gcc, g++, gfortran). Note that you can avoid all of this if you build HEASOFT using Mac's own Xcode gcc and g++, and only use Fink or MacPorts for gfortran.

You are going to have to get the corresponding Fink or MacPorts distribution of Python for both building and running PyXspec. The standard Xcode Python (in /usr/bin) will conflict with libraries pulled in by the Fink and MacPorts gcc.

To rebuild PyXspec using the Fink or MacPorts Python, first edit the file heasoft-<ver>/-Xspec/src/XSUser/Python/xspec/Makefile by adding definitions for PYTHON_INC and PYTHON_LIB that point to your Fink or Mac Python header and library files. For example if using the Fink Python v2.6 in its default location, you would insert the following in your Makefile, after the definition for HD_LIBRARY_ROOT and before the definition for HD_CXXFLAGS:

```
PYTHON_INC = /sw/include/python2.6

PYTHON_LIB = -L/sw/lib/python2.6/config # Note that this
must begin with '-L'
```

Then from the same directory containing the Makefile, do:

```
hmake clean

hmake

hmake install
```

3.4 Troubleshooting

If the HEASOFT configuration stage fails when it's processing PyXspec, it will just issue a warning and continue. Its failure should not affect the rest of the XSPEC and HEASO-FT build. Standard XSPEC will still be fully functional, but its Python interface won't be available.

The most likely cause of a PyXspec build failure is that the HEASOFT configuration script can't find a **python** executable and/or its accompanying library and header files. You should first check that the command "which python" can find an executable on your path. The configuration script first looks for **python**, which is normally a symbolic link to the version-specific executable. If it doesn't find that, it looks for **python2.7** down to **python2.3** in descending order.

Once it's found an executable, it looks for Python.h and libpython[m.n].so (or .dylib) in the directories ../include/python[m.n] and ../lib respectively, relative to the executable location. The configuration fails if either file is missing.

If you are running on **Mac OS X** and get a Python "ImportError" message containing such statements as **no suitable image found** and **mach-o, but wrong architecture**, it's likely you are running 32-bit mode Python while your default HEASoft build is now 64-bit. Make sure you are NOT still using the settings in Case 1 and 2 above for running 32-bit Python.

If you are running on a **Mac** and have built your HEASOFT installation with **all 3 compilers (gcc, g++, gfortran)** coming from **Fink** or **MacPorts**, AND you get a runtime error that begins with something like:

```
python(86419) malloc:  *** error for object 0x574b160:
pointer being freed was not allocated
```

then it likely means there's a conflict between your default Python distribution and the compiler libraries used to build PyXspec. Please see **Case 3** in the previous section for how to rebuild PyXspec with a Fink or MacPort distribution of Python.

4 A Tutorial - Quick Version

This assumes the user already has a basic familiarity with both XSPEC and Python. Everything in PyXspec is accessible by importing the package **xspec** into your Python script.

PyXspec can be utilized in a Python script or from the command line of the plain interactive Python interpreter. PyXspec does not implement its own command handler, so it is NOT intended to be run as the Python equivalent of a traditional interactive XSPEC session (which is really an *enhanced* interactive Tcl interpreter). In other words you launch an interactive PyXspec session with:

```
UNIX>python

>>> import xspec

>>>
```

rather than:

```
UNIX>xspec

XSPEC12>
```

Note that in all the tutorial examples the **xspec** package name qualifier is left off. You must either include the **xspec** qualifier:

```
s = xspec.Spectrum("file1.pha")
```

or use a variation of the Python `import` or `from...import` commands:

```
from xspec import *
s = Spectrum("file1.pha")
```

4.1 Jumping In - The Really Quick Version

A simple Xspec load-fit-plot Python script may look something like this:

```
#!/usr/bin/python
from xspec import *
Spectrum("file1.pha")
Model("wabs*pow")
Fit.perform()
Plot.device = "/xs"
Plot("data")
```

Keeping this template in mind, we'll proceed to fill in the details...

4.2 Terminology

This description uses the standard Python object-oriented terminology, distinguishing between **classes** and *objects*. **Class** is used when referring to the *type* or definition of an *object*. An *object* refers to a specific instance of a **class** and is normally assigned to a variable. For example a user may load 3 data files by creating 3 spectral data objects *s1*, *s2*, and *s3*, which are all instances of the class **Spectrum**.

The functions and stored data members that make up the definition of a **class** are referred to as **methods** and **attributes** respectively.

The term **Standard XSPEC** refers to the traditional ways of using XSPEC, either with a Tcl script or an interactive XSPEC session.

4.3 Getting Help

There are two ways to get help for programming with PyXspec classes. The first is by viewing the **Classes** section of this manual. The **Classes:Class List** subsection is particularly useful as an entry point, as it contains hyperlinks to descriptions of every PyXspec class that is part of the public interface. The second way is to call Python's built-in **help([*class*])** function from the interactive Python shell. Both methods will display essentially the same information, which originates in the class **docstrings** in the code files.

4.4 The 6 Global Objects

An XSPEC session fundamentally consists of loading data, fitting that data to a model, and plotting the results. To manage these operations, PyXspec offers the user 6 global objects: *AllChains*, *AllData*, *AllModels*, *Fit*, *Plot*, and *Xset*. Note that these are NOT the names of classes. They are instantiated *objects* of the **class** types shown in Table 1.

Object Name	Class	Role
AllChains	**ChainManager**	Monte Carlo Markov Chain container
AllData	**DataManager**	Container for all loaded data sets (objects of class **Spectrum**)
AllModels	**ModelManager**	Container for all **Model** objects
Fit	**FitManager**	Manager class for setting properties and running a fit
Plot	**PlotManager**	Manager class for performing XSPEC plots
Xset	**XspecSettings**	Storage class for Xspec settings

Table 1: Table 1. PyXspec global objects

PyXspec instantiates these objects immediately upon the importing of the **xspec** package. You cannot create any other objects of these class types, as they each allow only 1 instance of their type. (They are **singletons** in the language of design patterns.)

Operations involving these should ALWAYS be performed through the objects and NOT their class names. These class names should never appear in your code.

4.5 Loading And Removing Data

Spectral data files can be loaded in several ways. You can create an object of the **Spectrum** class by passing it the data file name:

```
s1 = Spectrum("file1.pha")
```

which also adds the new object *s1* to the *AllData* container. Or you can simply add the new file directly to the container without retrieving a **Spectrum** object:

```
AllData += "file1.pha"
```

Later you can always obtain a **Spectrum** object reference to any of the loaded spectra by passing *AllData* an integer:

```
s2 = AllData(2) # s2 is a reference to the 2nd loaded
spectrum
```

For more complicated data loading, you have access to the same functionality in -Standard XSPEC's **data** command. Simply pass a string to the *AllData* object's __-call__ method:

```
AllData("file1 file2 2:3 file3")
```

Note that only the last example allows you to assign multiple data groups, the 3rd spectrum being assigned to data group 2. Also note that in the last example any previously loaded data sets are removed, thus reproducing the behavior of Standard XSPEC's **data** command.

Other ways of removing Spectrum objects (ie. data sets) from the container:

```
AllData -= 3 # Removes the 3rd Spectrum object (the spectrum
with index number 3) from the container.
```

```
AllData -= s1 # Removes the Spectrum object s1.
```

```
AllData -= "*" # Removes all Spectrum objects.
```

```
AllData.clear() # Removes all Spectrum objects.
```

You can check the current state of the *AllData* container at any time by doing:

```
AllData.show()
```

Similarly, to view information about a single **Spectrum** object:

```
s2.show()
```

4.6 Defining Models

The basic way of defining an XSPEC model is to create an object of the PyXspec class **Model**. Simply pass in a string containing a combination of 1 or more XSPEC model **components**. Since this uses the same syntax as Standard XSPEC's **model** command, component abbreviations are allowed:

```
m1 = Model("wa*po + ga")
```

and to see a complete listing of available XSPEC model components, do:

```
Model.showList()
```

When you define a model like this, PyXspec also automatically adds the new object to the global *AllModels* container. If the model is applied to multiple data groups, object copies are added to the container for each data group.

Similar to the case of spectral data, you can also load models directly into the global container:

```
# Another way to define a new model and create an object
for each data group.
AllModels += "wa*po + ga"
# Retrieve the model object assigned to data group 2.
m2 = AllModels(2)
# Various ways to remove all model objects from the container.
AllModels.clear()
AllModels -= "*"
```

To display models and their parameters:

```
# This displays all parameters in all model objects:
AllModels.show()
# While this displays just parameters 1,2,3 and 5:
AllModels.show("1-3, 5")
# This displays a single model object:
m2.show()
```

For defining mulitple (or named) models and assigning multiple sources, please see the Extended Tutorial section.

4.6.1 Component and Parameter Objects

Model objects contain **Component** objects and **Component** objects contain **Parameter** objects. There are several ways to access and set components and parameters individually (and if you want to change many parameter values at once, it may be faster to use the Model or *AllModels* **setPars** methods described in the next section). Examples of individual Component and Parameter object access:

```
# Component objects are accessible-by-name as Model object
attributes*:
comp1 = m1.wabs
comp2 = m1.powerlaw
# Parameter objects are accessible-by-name as Component
object attributes:
par4 = m1.gaussian.LineE
# ...and we can modify their values:
```

```
par4.values = 3.895
m1.wabs.nH = 5.0
comp2.PhoIndex = 1.5
# Can also get a Parameter object directly from a Model,
without going through a Component.
# Just pass the Model the Parameter index number:
par5 = m1(5)
# Examples of numerical operations allowed with Parameter
objects:
par4 += 0.75
par4 *= 2.0
y1 = m1.wabs.nH*100.0
y2 = par4 + par5
```

(∗)For models with duplicate copies of components, see the Extended Tutorial for accessing Component objects by name.

Note that in the above examples, only the parameter's *value* is being accessed or modified. To change all or part of its FULL list of settings including auxiliary values: *value, fit delta, min, bot, top, max*, you can set its **values** attribute to a tuple or list of size 1-6:

```
par4.values = 4.3, .01, 1e-3
par4.values = [4.3, .01, 1e-3, 1e-2, 100, 200]
```

Or for greater flexibility you can set it to a string using Standard XSPEC's **newpar** command syntax:

```
# This allows you to set new values non-consecutively.
par4.values = "1.0, -.01,,,,150"
```

A quick way to freeze or thaw a parameter is to toggle its **frozen** attribute:

```
par4.frozen = False
par5.frozen = True
```

To link a parameter to one or more others, set its **link** attribute to a link expression string as you would have with the **newpar** command. To remove the link, set **link** to an empty string or call the parameter's **untie** method.

```
par5.link = "2.3 * 4" # Link par 5 to par 4 with a multiplicative
constant.
par5.link = "" # Removes the link.
par5.untie() # Also removes the link.
```

Also ALL linked parameters in a model object can be untied with a single call to the **Model** class **untie** method.

To display a parameter's full set of values (including auxiliary values), just print its **values** attribute:

```
>>> print par4.values
[6.5, 0.05, 0.0, 0.0, 1000000.0, 1000000.0]
>>>
```

4.6.2 Setting Multiple Parameters At A Time

You can set multiple parameter values with a single call using the **Model** or *AllModels* **setPars** methods. This may be considerably faster than setting parameters one at a time through the individual **Parameter** objects as shown in the previous section. With **setPars**, the model will be recalculated just ONCE after all the changes have been made. But when setting through individual **Parameter** objects, the model will be recalculated after EACH parameter change.

```
# For Model object m1, supply 1 or more new parameter
values in consecutive order:
m1.setPars(2.5, 1.4, 1.0e3) # This changes pars 1, 2, and
3.
# Can also change paramater auxiliary values by passing a
string using the same
# syntax as with Standard XSPEC's newpar command:
m1.setPars(.95, "1.8„-5,-4,10,10")
# Now set parameters NON-CONSECUTIVELY by passing a -
Python dictionary object.
# This example changes pars 1, 2, 4, and 6:
m1.setPars(.95, 1.2, {4:9.8, 6:2.0})
```

Parameters can also be initialized by passing values to the **Model** object constructor. You do this by setting the Model constructor's setPars keyword argument to a tuple, list, or dictionary (or just a single value or string if only setting the first parameter):

```
# Supply values for parameters 1 and 3, use defaults for
the rest.
m = Model("wa*ga", setPars={1:1.5, 3:.2})
# Supply values for 1 and 2, use defaults for the rest.
m = Model("wa*ga", setPars=(1.5, 0.7))
```

```
# Supply value only for 1.
m = Model("wa*ga", setPars=1.5)
```

Finally, if you wish to set multiple parameters that belong to **different** model objects, you must use the *AllModels* container's setPars method. This follows the same syntax rules as the single Model setPars, except that you also supply the Model objects as arguments:

```
# Change pars 1 and 3 in m1, and pars 1 and 2 in m2:
AllModels.setPars(m1, {1:6.4, 3:1.78}, m2, 3.5, 0.99)
```

4.7 Fitting

Once data and models are loaded, fitting is performed by calling the **perform** method of the *Fit* global object:

```
Fit.perform()
```

Some of the more frequently modified fit settings are the type of statistic to minimize and the maximum number of fit iterations to perform. These settings are attributes of *Fit*:

```
Fit.nIterations = 100
Fit.statMethod = "cstat"
Fit.statMethod = "chi"
```

Please see the class reference guide and the Extended Tutorial for *Fit*'s complete functionality.

To display the fit results at any time:

```
Fit.show()
```

4.8 Plotting

In Standard XSPEC, plot settings are adjusted using the **setplot** command while the plot is displayed through the **plot** command. In PyXspec, all plot settings and functionality is handled through the global *Plot* object. A device must be set before any plots can be displayed, and this done through the **device** attribute:

```
Plot.device = "/xs"
```

The device can also be set to print to an output file in several formats. The list of possible devices is given by the **cpd** command in the Standard XSPEC manual.

A typical setting to adjust is the X-axis units. You can choose to plot channel numbers, or select from various energy and wavelength units. The strings can also be abbreviated. Examples:

```
Plot.xAxis = "channel"

Plot.xAxis = "MeV"

Plot.xAxis = "Hz"

Plot.xAxis = "angstrom"
```

The displays of individual additive components or background spectra is toggled by setting their attributes to a bool:

```
Plot.add = True

Plot.background = False
```

Similarly log/linear settings for data plots (when using energy or wavelength units):

```
Plot.xLog = True

Plot.yLog = False
```

The current plot settings are displayed with:

```
Plot.show()
```

To actually display a plot, send 1 or more string arguments to the *Plot* __call__ method:

```
# Single panel plots

Plot("data")

Plot("model")

Plot("ufspec")

# Multi panel plots

Plot("data chisq")

Plot("data","model","resid")

# Call Plot with no arguments to repeat the previously
entered Plot command

Plot()
```

After displaying a plot, you can get an array of the plotted values by calling one of Plot's retrieval methods. All of these functions take an optional plot group number argument for the case of multiple plot groups, and all return the plot values in a Python list.

```
Plot("data")

xVals = Plot.x()

yVals = Plot.y()

yVals2 = Plot.y(2) # Gets values for data in the second
plot group

modVals = Plot.model()
```

```
# To get a background array, Plot.background must be set
prior to plot
Plot.background = True
Plot("data")
bkg = Plot.backgroundVals()
# Retrieve error arrays
xErrs = Plot.xErr()
yErrs = Plot.yErr()
```

5 A Tutorial - Extended Version

This assumes the user is familiar with the basics of PyXspec as explained in the Quick Tutorial.

5.1 Contents

5.2 Data

5.2.1 Background, Response, and Arf

When a **Spectrum** object is created from a spectral data file, PyXspec also reads the file's BACKFILE, RESPFILE, and ANCRFILE keywords and will load the corresponding background, response, and arf files. The spectrum's **Background** and **Response** objects are then available as attributes of the **Spectrum** class, while the arf file name becomes an attribute of the **Response** class:

```
s1 = Spectrum("file1")

b1 = s1.background

r1 = s1.response

arfFileName = r1.arf
```

Note that you never create **Background** and **Response** objects directly. They are accessible only through the **Spectrum** class attributes.

These attributes may also be used to add, change, or remove auxiliary files to an existing **Spectrum** object:

```
# Add or replace files:

s1.background = "newBackground.pha"

s1.response.arf = "newArf.pha"

# Removal examples:

s1.response = None

s1.background = ""
```

Background and **Spectrum** store their original file names in their **fileName** attribute. This means that while you SET the Spectrum.background object by assigning it a file name (as shown above), to GET the file name you must access its **fileName** attribute:

```
bkgFileName = s1.background # Wrong!!!  This returns the
entire Background object, not a string.
```

```
bkgFileName = s1.background.fileName # Correct
```

Response stores its RMF and optional ARF file names in its **rmf** and **arf** attributes respectively:

```
rmfFileName = r1.rmf
```

```
arfFileName = r1.arf
```

Background objects have some of the same attributes as **Spectrum** objects, such as **areaScale**, **exposure**, **energies**, and **values**. The **Spectrum** object's **values** array (actually a tuple) does NOT include contributions from the background. Those are stored separately in the associated **Background** object. Please see the **Classes** reference guide or call the Python help function for the full class descriptions.

The **Spectrum** class also provides a **multiresponse** array attribute for assigning multiple detectors (or sources) to a spectrum. The standard 0-based Python array indices corresponding to the 1-based XSPEC source numbers:

```
# Set a response for source 2

s1.multiresponse[1] = "resp2.rsp"

# Get the response object for source 2

r2 = s1.multiresponse[1]

# Remove the response from source 2

s1.multiresponse[1] = None

# This is the same as doing s1.response = "resp1.rsp"

s1.multiresponse[0] = "resp1.rsp"
```

The rule is: when doing single-source analysis (typical of most XSPEC sessions) use the **response** attribute, otherwise use the **multiresponse** array.

5.2.2 Ignore/Notice

To ignore channels for a SINGLE spectrum, call the **Spectrum** object's **ignore** method passing a string following the same syntax as for Standard XSPEC's **ignore** command:

```
s1.ignore("20-30 50-**")

s1.ignore("**-5")
```

Similarly, to notice channels in a single spectrum:

```
s1.notice("10-30,80-**")
```

```
s1.notice("all")
```

As with Standard XSPEC, if the **x-axis** plot units are set to energies or wavelengths, **ignore** and **notice** will accept floating-point input assumed to be in those same units:

```
Plot.xAxis = "nm"
```

```
# Ignore channel bins corresponding to 15.0 to 20.0 nm
wavelengths:
```

```
s1.ignore("15.-20.")
```

The currently noticed channel ranges are displayed for each spectrum in the `AllData.show()` output. You can also get a list of the individual noticed channel numbers from **Spectrum**'s **noticed** attribute:

```
>>> s1.noticed
```

```
[3,4,5,7,8,10]
```

To apply **ignore** and **notice** commands to ALL loaded spectra, call the methods from the global *AllData* object. To apply to a subset of loaded spectra, add a range specifier to the left of the colon:

```
# These apply to all loaded spectra
```

```
AllData.ignore("100-120, 150-200")
```

```
AllData.notice("all")
```

```
AllData.ignore("bad")
```

```
# These apply to a subset of loaded spectra
```

```
AllData.ignore("1-3:  60-65")
```

```
AllData.notice("2-**:50-60")
```

5.3 Models

5.3.1 Model With Multiple Data Groups

When a model is defined and spectra are assigned to multiple data groups, PyXspec will generate a **Model** object copy for each data group (assuming the spectra also have responses attached). So if:

```
m1 = Model("wa*ga")
```

```
AllData("file1 2:2 file2")
```

then there are 2 **Model** objects for the model definition wabs$*$gaussian. The variable *m1* is set to the object belonging to data group 1, and to get the object for data group 2 do:

```
m2 = AllModels(2)
```

m1 and *m2* will each have the same set of **Component** and **Parameter** objects.

Parameters can be accessed directly by index from the Model objects, and these indices are numbered from 1 to nParameters for ALL data group copies. So for the "wa∗ga" example above:

```
p = m1(2) # Returns the 2nd ('LineE') parameter from the
model for data group 1.

p = m2(2) # Returns the 2nd ('LineE') parameter from the
model for data group 2.

p = m2(6) # Wrong!!
```

5.3.2 Defining Multiple Models

Beginning with XSPEC12, it became possible to assign multiple sources to spectra, and each source may have its own model function definition. To keep track of multiple model definitions, XSPEC requires that you assign them names. In PyXspec, the model name and source number are supplied as additional arguments to the **Model** __init__ function:

```
# Define a model named "alpha" assigned to source 1

m_1_1 = Model("wa*po","alpha")

# Define a model named "beta" assigned to source 2

m_2_1 = Model("const*bbody","beta",2)

# (In both of these cases, the returned object belongs to
data group 1)
```

[As with Standard XSPEC, to define a model for source numbers $>$ 1 you first must load a detector response for the source. See "Background, Response, and Arf" in the previous section.]

Note that in all previous examples in this tutorial, we have been using unnamed models which were assigned to source 1. Named models and source numbers may also be defined directly into the *AllModels* container by passing in a tuple:

```
# Define a model named "defn1" assigned to source 1

AllModels += ("wa*po", "defn1")

# Define a model named "defn2" assigned to source 2

AllModels += ("const*bbody", "defn2", 2)

# This replaces "defn1" with an unnamed model for source
1
```

```
AllModels += "wa*gaussian"
```

and from which **Model** objects can be retrieved:

```
# Get the "defn2" Model object for data group 1
m_2_1 = AllModels(1,"defn2")
# ...and for data group 2
m_2_2 = AllModels(2,"defn2")
```

To view all current source number and model assignments, see the *AllModels.sources* attribute, which displays a dictionary of the [source number]:[model name] pairs.

To remove model definitions:

```
# Remove all data group copies of "defn2"
AllModels -= "defn2"
# Remove all data group copies of the unnamed model (defined
above as "wa*gaussian")
AllModels -= ""
# Remove all copies of ALL model definitions
AllModels.clear()
```

5.3.3 Component And Parameter Access Part 2

When PyXspec constructs a **Model** object, it immediately adds to it an attribute of type **Component** for every component in the model expression. The attribute has the same (full) name as the component in the original expression, allowing you to access it as:

```
m = Model("wa*pow")
c2 = m.powerlaw
```

However when a model contains multiple copies of the same component, this type of access becomes ambiguous. So to distinguish among copies, for any component making its 2nd or more appearance (from left to right), PyXspec will append "_n" to the attribute name where n refers to the component's position in the expression (again from left to right). Or to put it more simply:

```
m = Model("wa*po + po")
# This gets the leftmost powerlaw component
pow1 = m.powerlaw
# This gets the rightmost, which is the 3rd component in
the expression.
pow2 = m.powerlaw_3
```

The **Model** object also stores an attribute which is a just a list of the names of its constituent **Component** attributes:

```
>>> m.componentNames
['wabs', 'powerlaw', 'powerlaw_3']
```

This may be useful for example if writing a loop to access each of a model's components. Similarly **Component** objects have a **parameterNames** attribute, listing the names of their constituent **Parameter** attributes:

```
>>> m.powerlaw.parameterNames
['PhoIndex', 'norm']
```

5.3.4 Gain Parameters (Response Models)

Response Models differ from the regular kind in that they act on a Response rather than directly calculate a flux. At present there is only one kind of Response Model in Xspec, and this is **gain**. **gain** is a built-in attribute of all **Response** objects, and is of the class type **RModel**. It has 2 parameters for adjusting the energies of a Response: **slope** and **offset**. Gain parameters are initially off by default, but may be turned on simply by setting either one. For example:

```
s = Spectrum("file1")
# The spectrum's response has a gain attribute that is
not in use,
# which is the equivalent of having a slope fixed at 1.0
and offset = 0.0.
r = s.response
# Setting either the slope or offset turns the gain on
for this response.
# Both slope and offset will now be fit parameters.
r.gain.slope = 1.05
# The previous setting leaves the offset at 0.0.  Now
we'll change it.
r.gain.offset = .05
# You can set slope and offset at the same time using -
Response's setPars method.
r.setPars(.99, .03)
```

slope and **offset** are Parameter objects and therefore have the same interface as regular model parameters:

```
# Modify the parameter's auxilliary values
r.gain.offset = ".08,,.01,.01,.5,.5"
# Set a parameter link
r.gain.offset.link = ".005*1"
```

To remove the response fit parameters and return the Response back to its original state, call the **gain.off()** method:

```
# This deletes the slope and offset parameters.
# Any references to them become invalid.
r.gain.off()
```

5.3.5 Flux Calculations

To perform a Standard XSPEC **flux** or **lumin** calculation, call the *AllModels* methods **calcFlux** or **calcLumin** respectively:

```
AllModels.calcFlux(".3 1.0")
AllModels.calcFlux(".1 10.0 err")
AllModels.calcLumin(".1 10.  .05 err")
```

As in Standard XSPEC the results will be stored with the currently loaded spectra:

```
>>> s1 = AllData(1)
>>> s1.flux
(5.7141821510911499e-14, 0.0, 0.0, 4.0744161672429196e-05,
0.0, 0.0)
>>> s1.lumin
(30.972086553634504, 0.0, 0.0, 0.056670019567301853, 0.0,
0.0)
```

unless there are no spectra, in which case the results are stored with the model object:

```
>>> AllModels(1).flux
(5.6336924399373855e-10, 0.0, 0.0, 0.05929616315253175,
0.0, 0.0)
```

5.3.6 Local Models in C/C++/Fortran

In Standard XSPEC, local model libraries are built with the **initpackage** comamnd and then loaded with **lmod**. The *AllModels* container supplies both of these functions for doing the same thing in PyXspec:

```
AllModels.initpackage("myLocalMods","lmodel.dat")
```

```
AllModels.lmod("myLocalMods")
```

By default this looks in the directory set by the LOCAL_MODEL_DIRECTORY variable in your ∼/.xspec/Xspec.init start-up file. You can override this by giving these functions an absolute or relative path as a dirPath keyword argument (see the Class guide for details).

5.3.7 Local Models in Python

You can also write model functions in Python and insert them into the XSPEC library with the *AllModels* **addPyMod** method. You simply define a function with 3 arguments for energies, parameters, and flux. For example a powerlaw model function might look like:

```
def lpow(engs, params, flux):
   for i in range(len(engs)-1):
      pconst = 1.0 - params[0]
      val = math.pow(engs[i+1],pconst)/pconst - math.pow(engs[i],pconst)/pconst
      flux[i] = val
```

XSPEC will pass **tuples** containing the energy and parameter values to your function. For the flux array, it will pass a **list** pre-sized to nE-1, where nE is the size of the energies array. Your model function should fill in this list with the proper flux values. (For additional optional arguments to your model function, please see the documentation for the **addPyMod** function.)

The second thing you must define is a **tuple** containing the parameters' information strings, one string for each parameter in your model. This is equivalent to the parameter strings you would define in a 'model.dat' file when adding local models in standard XSPEC, and it requires the same format. (See Appendix C of the XSPEC manual for more details.) So with the powerlaw function above which takes just 1 parameter, you might define a tuple as:

```
powInfo = ("phoIndex \"" 1.1 -3.  -2.  9.  10.  0.01",)
```

Note the need for the trailing comma when there's just one parameter string. This is to let Python know that powInfo is a tuple type and not a string.

Once you've defined your function and parameter information, simply call:

```
AllModels.addPyMod(lpow, powInfo, 'add')
```

The 3rd argument tells XSPEC the type of your model function ('add', 'mul', or 'con'). After this call your function will be added to the list of available model components, which you can see by doing 'Model.showList()'. Your model will show up with the same name as your original Python function ('lpow'), and is ready for use in future Model definitions.

5.4 Fitting

5.4.1 Error

The **error** command is implemented through *Fit*, and the results are stored with the chosen **Parameter** object(s). The **error** attribute stores a tuple containing the low and high range values for the parameter, and the 9-letter status string to report problems incurred during the error calculation.

```
# Estimate the 90% confidence range for the 4th parameter
>>> Fit.error("2.706 4")
>>> par4 = AllModels(1)(4)
>>> par4.error
(0.11350354517707145, 0.14372981075906774, 'FFFFFFFFF')
```

5.4.2 Query

During a `Fit.perform()` operation, the default is to query the user whenever the fit has run the maximum number of iterations, as set by the `Fit.nIterations` attribute. You can change this behavior with the **query** attribute:

```
# When nIterations is reached, continue the fit without
stopping to query.
Fit.query = "yes"
# Stop fit at nIterations and do not query.
Fit.query = "no"
# Query the user when nIterations is reached.
Fit.query = "on"
```

5.4.3 Steppar

The Standard XSPEC **steppar** command is also implemented through the global *Fit* object. You supply it with a string following the same **steppar** command syntax rules. For example:

```
# Step parameters 1 and 2 through the given range values
# over a 10x10 2-D grid.
Fit.steppar("1 20.  30.  10 2 .05 .08 10")
```

5.5 Fakeit

PyXspec provides access to standard XSPEC's **fakeit** command, which is for creating spectra with simulated data. It is called through the *AllData* **fakeit** method:

```
AllData.fakeit(nSpectra=1, settings=None, applyStats=-
True, filePrefix="")
```

NOTE: If `AllData.fakeit` is run when spectra are currently loaded, it will follow the same rules as the standard XSPEC **fakeit** function: It will REMOVE ALL pre-existing spectra and replace each one with a simulated spectrum (even if nSpectra is less than the number originally loaded).

As those familiar with standard **fakeit** know, the user is normally prompted for quite a bit of additional information needed to generate the fakeit files. However the goal here is to have NO additional prompting, and that requires that all information must be entered as arguments to the *AllData* fakeit method call. This is done by passing objects of the **FakeitSettings** class to `AllData.fakeit`, as we'll show further below.

NOTE: Unless stated otherwise, assume all spectra are OGIP **type-1** (1 spectrum per file).

For the simplest of cases, you don't need to create any **FakeitSettings** objects. Just pass in the number of fake spectra you'd like to create:

```
# Create 3 fake spectra using only default settings.
```

```
AllData.fakeit(3)
```

The fakeit function will then create a default **FakeitSettings** object for each of the 3 spectra. By default, a **FakeitSettings** object will have empty strings for all of its attributes, and these are handled differently depending on whether the fake spectrum is replacing a currently loaded spectrum, or creating one from scratch.

5.5.1 From Existing Spectra

When replacing an existing spectrum, **FakeitSettings** attributes with empty strings will simply take their value from the original spectrum. Also note that the **response** and **arf** settings for the original spectrum CANNOT be modified for the fakeit spectrum. If a name is filled in for either of these attributes, it will be ignored. If you wish to modify these, you can make the change to the original spectrum prior to calling fakeit. [The one exception is when the original spectrum has no response, in which case the response attribute MUST be filled in.] If the **fileName** attribute is empty, XSPEC will generate a default output name derived from the original file name.

5.5.2 From Scratch

When creating from scratch, an empty string implies "none" for the arf and background, 1.0 for exposure and correction, and XSPEC's default dummy response for the response attribute. If the fileName attribute is empty, XSPEC will generate a default output file name based on the response name, and it will include an auto-incremented index to prevent multiple output files from overwriting each other.

5.5.3 FakeitSettings Objects

To create a fake spectrum with anything other than default settings, you must supply a **FakeitSettings** object for that spectrum. The **FakeitSettings** attributes are: response, arf, background, exposure, correction, backExposure, and fileName. All are string types, though exposure, backExposure, and correction can also be entered as floats. Attributes can be set upon object construction, or anytime afterwards:

```
fs1 = FakeitSettings("response1.rsp", exposure = 1500.0)

fs1.background = "back1.pha"
```

A new FakeitSettings object can also be made by copying an existing one:

```
fs2 = FakeitSettings(fs1)
```

And now pass the objects to the fakeit method, either in a list, dictionary, or as a single object:

```
# Apply settings to fakeit spectra 1 and 2:

AllData.fakeit(2,[fs1,fs2])

# Apply setting to fakeit spectrum 1, use defaults for
spectrum 2:

AllData.fakeit(2, fs1)

# Apply settings to fakeit spectra 2 and 4, use defaults
for 1 and 3:

settingsDict = {2:fs1, 4:fs2}

AllData.fakeit(4, settingsDict)

# Create 4 fakeit spectra from the same settings object:

settingsList = 4*[fs1]

AllData.fakeit(4, settingsList)
```

The remaining 2 arguments to the `AllData.fakeit` function are for choosing whether to apply statistical fluctuations (default = True), and whether to add an optional prefix string to the names of all output files.

5.5.4 OGIP Type-2 Files

With **OGIP type-2** files, multiple spectra may be placed in a single file. The important thing to recognize when generating type-2 fakeit files is that the **exposure, correction, backExposure,** and **fileName** attributes apply to the output **files** and not the individual spectra. Therefore these settings will be ignored for all but the first spectrum in a file. For example:

```
# Start with 4 spectra loaded, in 2 type-2 files:

AllData("myDataFile1.pha{1-2} myDataFile2.pha{7-8}")

# Create settings for the 4 fake spectra that will be
generated from these:

fs1 = FakeitSettings(background="back1.pha", exposure=250.)

# The exposure setting in fs2 will be ignored!!!

fs2 = FakeitSettings(background="back2.pha", exposure
100.)

fs3 = FakeitSettings(fileName="myFakeitFile_2.pha")

fs4 = FakeitSettings(fs3)

# The following change will be ignored!!!

fs4.fileName = "myFakeitFile_3.pha"

# Now generate the fakeit files:  AllData.fakeit(4, [fs1,fs2,fs3,fs4])
```

The above will generate 4 fakeit spectra, placed in 2 type-2 files. The exposure setting for spectrum 2 and the fileName setting for spectrum 4 will be ignored. Those values are only set by spectra 1 and 3.

For more fakeit details and examples, please check:

```
>>>help(FakeitSettings)

>>>help(DataManager.fakeit)
```

5.6 Monte Carlo Markov Chains (MCMC)

All MCMC operations are handled either by objects of class **Chain**, or the global *All-Chains* container object. To create a new chain based on the current fit parameters, simply create a **Chain** object by passing it an output file name:

```
c1 = Chain("chain1.fits")
```

The above call creates the file "chain1.fits", performs an MCMC run using the default *burn, fileType, length, proposal, rand, and temperature* values, and automatically places the new object in the *AllChains* container. These default settings are stored as attributes of *AllChains*:

```
# Ensure that new chains will burn the first 100 iterations,
will

# have length 1000, and will use the proposal "gaussian
fit"

AllChains.defBurn = 100

AllChains.defLength = 1000

AllChains.defProposal = "gaussian fit"

c2 = Chain("chain2.fits")
```

You can also override the *AllChains* default settings by passing additional arguments to **Chain** upon construction:

```
# Length will be 2000 for this chain, use defaults for
all other settings.

c3 = Chain("chain3.fits", runLength = 2000)
```

The new chain objects will then store their own settings as attributes:

```
>>>c2.burn

100

>>>c2.runLength

1000

>>>c3.runLength

2000
```

All of a chain object's attributes will be displayed when calling its `show()` method.

To append a new run to an existing chain object, call the object's `run()` method. - The appending run will use the object's current attribute settings, and not the *AllChains* default settings:

```
# This will append a run of length 3000 to the c3 chain
object, and with a

# Metropolis-Hastings temperature of 50.0:

c3.runLength = 3000

c3.temperature = 50.0

c3.run()

>>> c3.totalLength

5000
```

To **overwrite** rather than append to an existing chain object, call run with its append argument set to False:

```
# This erases the results of any previous runs for object
c3.

c3.run(False)

>>> c3.totalLength

3000
```

New chains are loaded into *AllChains* by default, but you can unload or reload them using the *AllChains* arithmetic operators:

```
# Chain c2 may be unloaded by passing its chain index
number

AllChains -= 2

# OR by passing the object itself

AllChains -= c2

# 2 ways to remove ALL chains

AllChains -= '*'

AllChains.clear()

# Reload an existing chain object

AllChains += c2

# Load a chain from an existing chain file

AllChains += "earlierChain.fits"

# Create a new chain, to be stored in file "chain4.fits"

AllChains += "chain4.fits"
```

As with Standard XSPEC, unloading a chain will leave the chain's file intact. It merely removes the chain from XSPEC's calculations. To display information about the currently loaded chains, call `AllChains.show()`.

You may also get a chain object from the container at any time by passing it an index number:

```
# Retrieve a chain object for the 4th chain in the container

c4 = AllChains(4)
```

5.7 Plotting

All of the plotting options available in Standard XSPEC's **setplot** command are now implemented as attributes of the *Plot* object. Some of these are mentioned in the Quick Version of the tutorial, and please see the PlotManager class reference for the complete guide.

One setting of particular interest is the **commands** attribute. This is a tuple of user-entered **PLT** command strings which are added to XSPEC's auto-generated commands when performing a plot, and is modified through *Plot*'s **addCommand** and **delCommand** methods. For example, to enter a PLT command to place an additional label at the specified coordinates on the plot:

```
Plot.addCommand("label 1 pos 10 .05 \"Another Label"")
```

To view the currently loaded commands:

```
print Plot.commands
```

and to remove the 3rd command from the tuple:

```
Plot.delCommand(3)
```

5.8 XSPEC Settings

Most of the internal switches set through Standard XSPEC's **xset** command are now set through attributes of the global *Xset* object. Examples:

```
Xset.abund = "angr"
```

```
Xset.cosmo = "50 .5 0."
```

```
Xset.xsect = "bcmc"
```

Xset also provides the methods **addModelString** and **delModelString** to set the <string name>,<string value> pairs which are used by certain models. The <string name> argument is case-insensitive.

```
Xset.addModelString("APECROOT","1.3.1")
```

```
Xset.addModelString("APECTHERMAL","yes")
```

```
Xset.delModelString("neivers")
```

The entire collection of <name>,<value> pairs may be set or retrieved with the – `Xset.modelStrings` attribute:

```
# Replace all previous entries with a new dictionary
```

```
Xset.modelStrings = {"neivers":"1.1", "apecroot":"1.3.1"}
```

```
# Clear out all entries:
```

```
Xset.modelStrings = {}
```

`Xset.show()` will display all of the current settings including the current <string name>,<string value> pairs.

5.9 Logging And XSPEC Output

The *Xset* object provides attributes and methods for controlling output chatter level and for creating log files:

```
# Get/Set the console chatter level

ch = Xset.chatter

Xset.chatter = 10

# Get/Set the log chatter level

lch = Xset.logChatter

Xset.logChatter = 20

# Create and open a log file for XSPEC output

# This returns a Python file object

logFile = Xset.openLog("newLogFile.txt")

# Get the Python file object for the currently opened log

logFile = Xset.log

# Close XSPEC's currently opened log file.

Xset.closeLog()
```

5.10 Exceptions And Error Handling

PyXspec utilizes the standard Python **try**/**except**/**raise** mechanism for handling and reporting errors. In this early version, only exception objects of the class **Exception** are ever raised. In the future other (more specific) error classes may be used, but they should always be derived from **Exception**. So you can catch all PyXspec exceptions with code such as:

```
try:
    # Only 4 spectra are currently loaded
     s = xspec.AllData(5)
except Exception, msg:
     print msg
```

which will print the error message:

```
Error:  Spectrum index number is out of range:  5
```

PyXspec raises errors in a variety of situations, such as for invalid input argument syntax, or for input which is invalid within the context of the call (as in the example above). It can also raise exceptions if you try to rebind a class attribute when such modification is not permitted.

5.11 Adding Attributes To PyXspec Objects

A particularly novel feature of Python (in comparison with say C++) is that it allows you to create new attributes "on the fly". The attributes don't have to have been part of the original class definition:

```
class C:
    pass

x = C()
x.pi = 3.1416
```

The downside of course is that spelling or case sensitive errors become much harder to detect. For example, with PyXspec's *Plot* object:

```
Plot.yLog = True # Correct

Plot.ylog = True # Wrong!
```

In the second case, standard Python will simply add a new attribute named "ylog" to *Plot*, and this will have no effect on the actual plot since PyXspec is only looking at "yLog".

So operating under the assumption that this downside outweighs the benefits, we've decided to **disable** the ability to add new attributes to PyXspec class objects. A misspelling or case error will instead raise an **Exception** object. And since some users may genuinely wish to add their own attributes to PyXspec classes, this default behavior may be overridden by toggling the **Xset.allowNewAttributes** flag:

```
s = Spectrum("dataFile.pha")

s.myNewIndex = 10 # Error:  Will raise an exception

Xset.allowNewAttributes = True

s.myNewIndex = 10 # OK

.

.  # Can add new attributes to any PyXspec object,

.  # but attribute spelling errors will go undetected.

.
```

```
Xset.allowNewAttributes = False
```

5.12 Using With Other Packages

One of the primary benefits of PyXspec is that it makes it much easier to use XSPEC data and results in 3rd party packages. For example you can bypass XSPEC's built-in plotting functions in favor of a Python plotting library such as Matplotlib:

```
#!/usr/bin/python
from xspec import *
import matplotlib.pyplot as plt
# PyXspec operations:
s = Spectrum("file1.pha")
m = Model("wa*po")
Fit.perform()
# Plot using Matplotlib:
plt.plot(s.noticed, s.values, 'ro', s.noticed, m.folded(1))
plt.xlabel('channels')
plt.ylabel('counts/cm^2/sec/chan')
plt.savefig('myplot')
```

The above code produces a Matplotlib plot of the spectral data and folded model vs. channels (similar to what you get with Standard XSPEC's "plot data" command). - It makes use of the **Spectrum** object's **noticed** attribute to pass a list of the channel numbers, and the **values** attribute (a tuple) to pass the spectral data values in counts/cm^2/s. The folded model values are obtained as a list by calling the **Model** object's **folded** method with a spectrum number argument.

6 What's Missing

Python equivalents for these standard XSPEC commands are not yet implemented:

- hardcopy
- identify
- margin
- mdefine

- Tcl script commands: addline, lrt, modid, multifake, rescalecov, simftest, writefits

The following commands perform functions which are not applicable to the currently intended design and usage of PyXspec, and therefore are not likely to be implemented in the near future:

- addcomp
- autosave
- delcomp
- editmod
- save
- script

7 Class Index

7.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

8 Class Documentation

8.1 Background Class Reference

Public Member Functions

- def __init__

Public Attributes

- areaScale

  The Background area scaling factor (GET only).
- backScale

  The Background back scaling factor (GET only).
- exposure

  The exposure time keyword value [float] (GET only).
- fileName

  The spectrum's file name [string] (GET only).
- isPoisson

  Boolean flag, True if spectrum has Poisson errors (GET only).
- values

  Tuple of floats containing the background rates array in counts/cm^2-sec (GET only).
- variance

  Tuple of floats containing the variance of each channel (GET only).

8.1.1 Detailed Description

```
Background spectral data class.

Public instance attributes (implemented as properties):

   areaScale    -- The Background area scaling factor (GET only).
                   Either a single float (if file stores it as a keyword),
                      or a Tuple of floats (if file stores column).

   backScale    -- The Background back scaling factor (GET only).
                   Either a single float (if file stores it as a keyword),
                      or a Tuple of floats (if file stores column).

   exposure     -- The exposure time keyword value [float] (GET only).

   fileName     -- The spectrum's file name [string] (GET only).

   isPoisson    -- Boolean flag, True if spectrum has Poisson errors
                      (GET only).

   values       -- Tuple of floats containing the background rates array in
                      counts/cm^2-sec (GET only).

   variance     -- Tuple of floats containing the variance of each
                      channel (GET only).
```

8.1.2 Constructor & Destructor Documentation

8.1.2.1 def __init__ (*self*, *backTuple*, *parent*)

```
Construct a Background object.

Intended for creation by a Spectrum object only.
The parent arg should be the Spectrum object's self pointer.
```

8.1.3 Member Data Documentation

8.1.3.1 areaScale

The Background area scaling factor (GET only).

```
 Either a single float (if file stores it as a keyword),
    or a Tuple of floats (if file stores column).
```

8.1.3.2 backScale

The Background back scaling factor (GET only).

```
 Either a single float (if file stores it as a keyword),
    or a Tuple of floats (if file stores column).
```

8.1.3.3 exposure

The exposure time keyword value [float] (GET only).

8.1.3.4 fileName

The spectrum's file name [string] (GET only).

8.1.3.5 isPoisson

Boolean flag, True if spectrum has Poisson errors (GET only).

8.1.3.6 values

Tuple of floats containing the background rates array in counts/cm^2-sec (GET only).

8.1.3.7 variance

Tuple of floats containing the variance of each channel (GET only).

The documentation for this class was generated from the following file:

- spectrum.py

8.2 Chain Class Reference

Public Member Functions

- def __init__
- def run
- def show

Public Attributes

- burn

  The number of steps that will be thrown away prior to storing the chain [int].
- runLength

  The length of chain to be added during the next run [int].
- proposal

  The proposal distribution and source of covariance information to be used for the next run [string].
- rand

  Determines whether chain start point will be randomized (True) or taken from the current parameters (False).
- temperature

  The temperature parameter used in the Metropolis-Hastings algorithm for the proposal acceptance or rejection [float].
- fileName

Chain output file name.

- fileType

  Output format of the chain file [string].

- totalLength

  The cumulative length of the chain [int].

8.2.1 Detailed Description

```
Monte Carlo Markov Chain class.

Public instance attributes:

   GET-only attributes:

   fileName    -- Chain output file name.
   fileType    -- Output format of the chain file [string].
                     Will be either "fits" (the default), or "ascii".
   totalLength -- The cumulative length of the chain [int].
                     This will increase every time a run is performed.

   The following attribute settings will apply to the NEXT run for this
   chain.  The burn and rand settings are irrelevant if run is performing
   an appending operation.

   runLength   -- The length of chain to be added during the next run [int].
   proposal    -- The proposal distribution and source of covariance
                     information to be used for the next run [string].
                     Examples: "gaussian fit", "cauchy fit",
                               "gaussian chain", etc.
                     See the "chain" command in the standard XSPEC manual
                     for more information.
   temperature -- The temperature parameter used in the Metropolis-Hastings
                     algorithm for the proposal acceptance or rejection
                     [float].
   burn        -- The number of steps that will be thrown away prior to
                     storing the chain [int].
   rand        -- Determines whether chain start point will be randomized
                     (True) or taken from the current parameters (False).
```

8.2.2 Constructor & Destructor Documentation

8.2.2.1 def __init__ (*self*, *fileName*, *fileType* = None, *burn* = None, *runLength* = None, *proposal* = None, *rand* = None, *temperature* = None)

```
Construct a chain object, perform a run, and load into AllChains
      container.

The only required argument is fileName.  All other arguments will
take their default values from the current settings in the AllChains
container.
```

8.2.3 Member Function Documentation

8.2.3.1 def run (*self*, *append* = `True`)

```
Perform a new chain run, either appending to or overwriting an
      existing chain.

append -- If this is set to True the new run will be appended.
   If False, the new run will overwrite.  Note that the burn
   and rand settings do not apply when appending.
```

8.2.3.2 def show (*self*)

```
Display current settings of Chain object's attributes.
```

8.2.4 Member Data Documentation

8.2.4.1 burn

The number of steps that will be thrown away prior to storing the chain [int].

8.2.4.2 fileName

Chain output file name.

8.2.4.3 fileType

Output format of the chain file [string].

```
Will be either "fits" (the default), or "ascii".
```

8.2.4.4 proposal

The proposal distribution and source of covariance information to be used for the next run [string].

```
Examples: "gaussian fit", "cauchy fit",
          "gaussian chain", etc.
See the "chain" command in the standard XSPEC manual
for more information.
```

8.2.4.5 rand

Determines whether chain start point will be randomized (True) or taken from the current parameters (False).

8.2.4.6 runLength

The length of chain to be added during the next run [int].

8.2.4.7 temperature

The temperature parameter used in the Metropolis-Hastings algorithm for the proposal acceptance or rejection [float].

8.2.4.8 totalLength

The cumulative length of the chain [int].

```
 This will increase every time a run is performed.
```

The documentation for this class was generated from the following file:

- chain.py

8.3 ChainManager Class Reference

Public Member Functions

- def __init__
- def __call__
- def __iadd__
- def __isub__
- def clear
- def show
- def stat

Public Attributes

- defBurn

  Default burn length for new Chain objects (orig = 0).
- defFileType

  Default output file format (orig = "fits").
- defLength

  Default chain length (orig = 100).
- defProposal

  Default chain proposal (orig = "gaussian fit").
- defRand

Default randomization setting (orig = False).

- defTemperature

  Default chain temperature (orig = 1.0).

8.3.1 Detailed Description

```
Monte Carlo Markov Chain container.

This is a singleton - only 1 instance allowed

Public instance attributes:

   These are the values which will be used when creating new Chain objects,
   unless they are explicitly overridden as arguments to the Chain class
   constructor.  For more detail, see the descriptions for the
   corresponding attributes in the Chain class doc.

   defBurn         -- Default burn length for new Chain objects (orig = 0).
   defFileType     -- Default output file format (orig = "fits").
   defLength       -- Default chain length (orig = 100).
   defProposal     -- Default chain proposal (orig = "gaussian fit").
   defRand         -- Default randomization setting (orig = False).
   defTemperature  -- Default chain temperature (orig = 1.0).
```

8.3.2 Constructor & Destructor Documentation

8.3.2.1 def __init__ (*self*)

8.3.3 Member Function Documentation

8.3.3.1 def __call__ (*self, index*)

```
Get a Chain object from the AllChains container.

   index -- The index of a currently loaded chain file.  The list
      of currently loaded chains can be seen with the
      AllChains.show() method.  The valid range is:
      1 <= index <= nLoadedChains.

   Note that the returned Chain object's modifiable attributes will
   be initialized with the current AllChains def<attribute> settings.

   Example:
      # Load 2 chains from pre-existing files:
      AllChains += "chain1.fits"
      AllChains += "chain2.fits"
      # and get a Chain object for the 2nd chain:
      c2 = AllChains(2)
```

8.3.3.2 def __iadd__ (*self*, *chain*)

```
Load a pre-existing chain into the AllChains container.

Argument may be a currently existing chain object which had been
 unloaded earlier:
    AllChains += myChain1
the filename of an existing chain file:
    AllChains += "chainFile.fits"
or the filename of a new chain:
    AllChains += "newChainFile.fits" # This will also perform a chain
                             # run using the default settings.
```

8.3.3.3 def __isub__ (*self*, *chain*)

```
Unload one or more chain objects from container.

Argument may either be a chain object:
    AllChains -= myChain1
a filename:
    AllChains -= "chainFile.fits"
the chain's current index [int] in the AllChains container:
    AllChains -= 2
or a '*' to unload ALL chains (equivalent to AllChains.clear()):
    AllChains -= '*'
```

8.3.3.4 def clear (*self*)

```
Unload all chains from container
```

8.3.3.5 def show (*self*)

```
Display information for current attributes and loaded chains.
```

8.3.3.6 def stat (*self*, *parIdx*)

```
Display statistical information on a particular chain parameter.

   parIdx -- The parameter index number, including optional model
       name: [<modName>:]<idx>.  May be entered as a string
       or int (if no model name).
```

8.3.4 Member Data Documentation

8.3.4.1 defBurn

Default burn length for new Chain objects (orig = 0).

8.3.4.2 defFileType

Default output file format (orig = "fits").

8.3.4.3 defLength

Default chain length (orig = 100).

8.3.4.4 defProposal

Default chain proposal (orig = "gaussian fit").

8.3.4.5 defRand

Default randomization setting (orig = False).

8.3.4.6 defTemperature

Default chain temperature (orig = 1.0).

The documentation for this class was generated from the following file:

- chain.py

8.4 Component Class Reference

Public Member Functions

- def __init__
- def __setattr__

Public Attributes

- name

    The full name of the Component (get only).
- parameterNames

    List of Component's parameter names (get only).

8.4.1 Detailed Description

```
Model component class.

Public instance attributes:

   name            -- The full name of the Component (get only).
   <parameters>    -- Component contains an attribute of type Parameter for
                        every parameter in the component.  The attribute
                        name is the same as the parameter name in xspec.

   parameterNames  -- List of Component's parameter names (get only).
```

8.4.2 Constructor & Destructor Documentation

8.4.2.1 def __init__ (*self*, *compName*, *parNames*)

```
Component constructor.

Intended for creation by Model objects only.

compName   -- The full xspec component name.  This will also
        be the name of the attribute in the model object
parNames   -- List containing component's parameter names.
```

8.4.3 Member Function Documentation

8.4.3.1 def __setattr__ (*self*, *attrName*, *value*)

8.4.4 Member Data Documentation

8.4.4.1 name

The full name of the Component (get only).

8.4.4.2 parameterNames

List of Component's parameter names (get only).

The documentation for this class was generated from the following file:

- model.py

8.5 DataManager Class Reference

Public Member Functions

- def __init__
- def __call__
- def __isub__
- def __iadd__
- def clear

- def diagrsp
- def dummyrsp
- def fakeit
- def ignore
- def notice
- def removeDummyrsp
- def show

Public Attributes

- nGroups

  The number of data groups [int].
- nSpectra

  The number of loaded spectra [int].

8.5.1 Detailed Description

```
Spectral data container.

This is a singleton - only 1 instance allowed

Public instance attributes, GET only unless stated otherwise.

   nGroups  -- The number of data groups [int].
   nSpectra -- The number of loaded spectra [int].
```

8.5.2 Constructor & Destructor Documentation

8.5.2.1 def __init__ (*self*)

8.5.3 Member Function Documentation

8.5.3.1 def __call__ (*self, expr*)

```
DataManager get or set spectra.

Get:
  expr -- An integer referring to the spectrum index number.  Returns
    the spectrum, or raises an Exception if the integer is out
    of range.
Set:
  expr -- A string following the same syntax rules as Xspec's
    traditional "data" command handler.
```

8.5.3.2 def __iadd__ (*self*, *spectra*)

```
Add 1 spectrum to the data container.

spectra - the data filename string.
```

8.5.3.3 def __isub__ (*self*, *spectra*)

```
Remove 1 or all spectra from the data container.

spectra - either a single spectrum index number (int), a single
  Spectrum object, or the string "*" to remove all.
```

8.5.3.4 def clear (*self*)

```
Remove all spectra from the data container.
```

8.5.3.5 def diagrsp (*self*)

```
Diagonalize the current response matrix for ideal response.

All currently loaded responses will be replaced with diagonal
response matrices.  The energy range and channel binning
information are retained from the original response, as is the
effective area.  The channel values are mapped directly into the
corresponding energy ranges to simulate a detector with perfect
spectral resolution.

To remove diagonal responses and restore the originals, call the
AllData.removeDummyrsp() method.
```

8.5.3.6 def dummyrsp (*self*, *lowE =* None, *highE =* None, *nBins =* None, *scaleType =* None, *chanOffset =* None, *chanWidth =* None)

```
Create a dummy response and apply it to all spectra.

   Input arguments, all are optional:

     lowE  - Input response energy lower bound, in keV. [float]
     highE - Input response energy higher bound, in keV. [float]
     nBins - Number of bins into which the energy range is divided
        [int].
     scaleType - "log" or "lin" [string]
     chanOffset - Starting value of dummy channel energies. [float]
     chanWidth  - Energy width of the channel bins.  [float]
             If this is set to 0, the dummy response
             can only be used for evaluating model arrays,
             and not for fitting to spectra.
   Examples:

# All values are optional, use keywords to enter values
# non-consecutively.  Unspecified values revert to the
# current defaults.
AllData.dummyrsp(.3, 30., 100, chanWidth=.5)
AllData.dummyrsp(highE = 50.)
AllData.dummyrsp(.1,10.,100,"lin",.0, 1.0)

Initial defaults:  lowE = .1, highE = 50., nBins = 50, scaleType = "log"
   chanOffset = .0, chanWidth = .0
The defaults for lowE, highE, nBins, scaleType, and chanOffset will be
modified for each explicit new entry.  chanWidth always defaults to 0.

To remove dummy responses and restore actual responses (if any), call
the removeDummyrsp() method.

To apply a dummy response to just a single spectrum, use the
Spectrum.dummyrsp method.
```

8.5.3.7 def fakeit (*self*, *nSpectra* = `1`, *settings* = `None`, *applyStats* = `True`, *filePrefix* = `""`, *noWrite* = `False`)

```
Produce spectra with simulated data using XSPEC's fakeit command.

Note that if this method is run when spectra are currently loaded, it
will follow the same rule as the standard XSPEC fakeit function:
It will REMOVE ALL pre-existing spectra and replace each one with
a simulated spectrum (even if nSpectra is less than the number
originally loaded).

All arguments are optional:

  nSpectra   -- The number of fake spectra to produce.  [int]

        If there are nOrig pre-existing spectra loaded at the
        time this function is called and nSpectra < nOrig,
        nSpectra will be RESET to nOrig (see note above).

        If nSpectra == nOrig, then each of the fake spectra
        will use the settings from the respective original
        spectra for their defaults (see the FakeitSettings
        class description).

        If nSpectra > nOrig, then settings for the fake spectra
        numbered above nOrig will not be based on pre-existing
        spectra (if any).

  settings   -- A collection of 0 to nSpectra FakeitSettings objects,
        which may be entered as a list, a dictionary, a
        single FakeitSettings object, or None.

        If settings is a dictionary, the key,value pairs should
        be the spectrum index number (1 is lowest) and the
        FakeitSettings object.
        This function will match up FakeitSettings objects
        1-to-1 with the nSpectra fake spectra to be created.

        If user provides FEWER than nSpectra FakeitSettings
        objects, fakeit will generate the necessary additional
        objects with their default settings.

        If MORE than nSpectra FakeitSettings objects are
        provided, the extra objects will be ignored.

  applyStats -- If set to True, statistical fluctuations will be
        included in the generation of fake spectra.  [bool]

  filePrefix -- Optional string to attach as a prefix to default fakeit
        output file names.  Note that this only applies when
        using the default file names.  If a file name is
        explicitly entered in the FakeitSettings.fileName
        attribute, it will not make use of this.

  noWrite    -- If set to True, no fakeit output files will be generated.
        Default is False.  [bool]

Examples:

   # Assume no data is loaded, but a model is defined:

   AllData.fakeit()
   # Creates 1 fake spectrum using the default FakeitSettings object,
   # which has all input strings empty.  So it will use XSPEC's internal
   # dummy response and its output file name will be dummy_rsp_1.fak.

   # Now assume AllData contains 2 spectra PRIOR to running EACH of the
   # following commands, then:

   AllData.fakeit()
   # Creates 2 fake spectra with all settings (response, arf,
   # background, exposure, corrscale, backExposure, filenames) based
   # on the original spectra.  The original 2 spectra are removed from
   # AllData.

   AllData.fakeit(3)
   # Creates the first 2 spectra as above.  The 3rd fake spectrum is
   # based on the default FakeitSettings object and its output filename
   # will be dummy_rsp_3.fak

   fs = FakeitSettings(background="back1.pha", exposure=2000.0)
   sl = 3*[fs]
   AllData.fakeit(3, sl)
   # Same as above, but all 3 fake spectra will have a background file
   # based on back1.pha, and exposure time = 2000.0 sec.

   AllData.fakeit(3, sl, False, "my_fake_")
   # Same as above, but no statistical fluctuations will be applied to
   # fake spectra, and all output files will have the "my_fake_"
   # prefix attached.
   fs1 = FakeitSettings("resp1.rmf","arf1.pha",exposure=1500.)
   fs2 = FakeitSettings(fs1)
   fs2.response = "resp2.rmf"
   sd = {3:fs1, 5:fs2}
   AllData.fakeit(5, sd)
   # Creates 5 fake spectra.  The first 2 use the settings from the
   # originally loaded data.  Spectra 3 and 5 use the settings from
   # the fs1 and fs2 FakeitSettings objects, which differ only in their
   # response names.  Spectrum 4 uses the default FakeitSettings object.
```

8.5.3.8 def ignore (*self, ignoreRange*)

```
Apply an ingore channels range to multiple loaded spectra.

ignoreRange -- String specifying the spectra ranges and/or
         channel ranges to ignore, or "bad".
         This follows the same syntax as used in the standard
         Xspec "ignore" command, except that the spectrum range
         always defaults to ALL spectra.

         If the channel ranges are floats rather than ints,
         they will be treated as energies or wavelengths
         (depending on the Plot settings).
```

8.5.3.9 def notice (*self, noticeRange*)

```
Apply a notice channels range to multiple loaded spectra.

noticeRange -- String specifying the spectra ranges and/or channel
         ranges to notice.  This follows the same syntax as
         used in the standard Xspec "notice" command, except
         that the spectrum range always defaults to ALL spectra.

         If the numbers are floats rather than ints, they will
         be treated as energies or wavelengths (depending on
         the Plot settings).  If the string is "all", it will
         notice all channels in all spectra.
```

8.5.3.10 def removeDummyrsp (*self*)

```
Remove all dummy responses, restore original responses (if any).
```

8.5.3.11 def show (*self*)

```
Display information for all loaded spectra.
```

8.5.4 Member Data Documentation

8.5.4.1 nGroups

The number of data groups [int].

8.5.4.2 nSpectra

The number of loaded spectra [int].

The documentation for this class was generated from the following file:

- data.py

8.6 FakeitSettings Class Reference

Public Member Functions

- def __init__

Public Attributes

- response

  Name of detector response file to use for creating the fake spectrum.
- arf

  Name of optional arf to use with the response.
- background

  Name of optional background file to use when creating the fake spectrum.
- exposure

  The fake spectrum exposure time.
- correction

  Optional correction norm factor.
- backExposure

  Optional background exposure time modifier.
- fileName

  Optional fake spectrum output file name.

8.6.1 Detailed Description

```
Fakeit command settings class.

The AllData.fakeit function will apply 1 FakeitSettings object to every
fake spectrum that is to be created.  If the user does not explicitly
supply their own FakeitSettings objects, AllData.fakeit will create its
own as necessary, with default settings.
```

```
Public instance attributes [all are string types unless noted]:

   response   -- Name of detector response file to use for creating the
                    fake spectrum.

                 When a fake spectrum is based on a pre-existing spectrum
                    which already has a response, this should be left empty.
                    If a name is given it will be IGNORED.  However if the
                    pre-existing spectrum has no response, then this MUST be
                    filled.

                 If the fake spectrum is not based on an existing spectrum,
                    this may be filled or left empty.  If it is empty,
                    XSPEC will just use its built-in dummy response.

   arf        -- Name of optional arf to use with the response.  This is
                    ignored if no response is given.

   background -- Name of optional background file to use when creating the
                    fake spectrum.

                 If based on an original spectrum, leave this empty to use
                    the original spectrum's background settings.

   exposure   -- The fake spectrum exposure time.

   correction -- Optional correction norm factor.

   backExposure -- Optional background exposure time modifier.

                 For exposure and correction, if left empty fakeit will use
                    the values from the original spectrum, or 1.0 if not
                    based on an original spectrum.  Each of these may be
                    entered as a string or float.

   fileName   -- Optional fake spectrum output file name.

                 If left empty, fakeit will create a default file name
                    based on the original spectrum, or the response name
                    if no original spectrum.  In the latter case, the
                    default names will also have an incremented suffix to
                    prevent file overwriting.

  When writing to a multiple-spectrum output file (OGIP type-2), exposure,
  correction, backExposure, and fileName are applied to the entire file
  rather than a single spectrum.  Therefore entries for these attributes
  will be IGNORED for all but the first fake spectrum in a type-2 output
  file.
```

8.6.2 Constructor & Destructor Documentation

8.6.2.1 def __init__ (*self*, *response* = "", *arf* = "", *background* = "", *exposure* = "", *correction* = "", *backExposure* = "", *fileName* = "")

```
Create a FakeitSettings object.

All arguments are optional, and all may be entered as strings.
The exposure and correction arguments may also be entered as floats.

This can also create a new copy of a pre-existing FakeitSettings
object, in which case the pre-existing object should be the only
argument entered.

Examples:

   fs1 = FakeitSettings("resp1.pha", exposure=1500.0)
   # Reuse fs1's settings, but with a new fileName attribute:
   fs2 = FakeitSettings(fs1)
   fs2.fileName = "fakeit2.pha"
   # Now generate 2 fake spectra
   AllData.fakeit(2, [fs1, fs2])
```

8.6.3 Member Data Documentation

8.6.3.1 arf

Name of optional arf to use with the response.

This is

```
 ignored if no response is given.
```

8.6.3.2 backExposure

Optional background exposure time modifier.

```
 For exposure and correction, if left empty fakeit will use
    the values from the original spectrum, or 1.0 if not
    based on an original spectrum.  Each of these may be
    entered as a string or float.
```

8.6.3.3 background

Name of optional background file to use when creating the fake spectrum.

```
 If based on an original spectrum, leave this empty to use
    the original spectrum's background settings.
```

8.6.3.4 correction

Optional correction norm factor.

8.6.3.5 exposure

The fake spectrum exposure time.

8.6.3.6 fileName

Optional fake spectrum output file name.

```
If left empty, fakeit will create a default file name
   based on the original spectrum, or the response name
   if no original spectrum.  In the latter case, the
   default names will also have an incremented suffix to
   prevent file overwriting.
```

8.6.3.7 response

Name of detector response file to use for creating the fake spectrum.

```
When a fake spectrum is based on a pre-existing spectrum
   which already has a response, this should be left empty.
   If a name is given it will be IGNORED.  However if the
   pre-existing spectrum has no response, then this MUST be
   filled.

If the fake spectrum is not based on an existing spectrum,
   this may be filled or left empty.  If it is empty,
   XSPEC will just use its built-in dummy response.
```

The documentation for this class was generated from the following file:

- data.py

8.7 FitManager Class Reference

Public Member Functions

- def __init__
- def error
- def ftest
- def goodness
- def improve

- def perform
- def renorm
- def show
- def steppar
- def stepparResults

Public Attributes

- bayes

  Turn Bayesian inference on or off [string].
- covariance

  The covariance matrix from the most recent fit [tuple of floats] (GET only).
- criticalDelta

  Critical delta for fit statistic convergence [float].
- delta

  Set fit delta values to be proportional to the parameter value [float].
- dof

  The degrees of freedom for the fit [int] (GET only).
- method

  The fitting algorithm to use [string].
- nIterations

  The maximum number of fit iterations prior to query [int].
- query

  The fit query setting [string].
- statistic

  Fit statistic value from the most recent fit [float] (GET only).
- statMethod

  The type of fit statistic in use [string].
- statTest

  The type of test statistic in use [string].
- testStatistic

  Test statistic value from the most recent fit [float] (GET only).
- weight

  Change the weighting function used in the calculation of chi-sq [string].

8.7.1 Detailed Description

```
Xspec fitting class.

This is a singleton - only 1 instance allowed

Public instance attributes (implemented as properties):

   bayes         -- Turn Bayesian inference on or off [string].

                      Valid settings are "on", "off" (default), or "cons".
                      "cons" turns Bayesian inference on AND gives ALL
                      parameters a constant prior.  Priors can be set for
                      parameters individually through the Parameter object's
                      'prior' attribute.

   covariance    -- The covariance matrix from the most recent fit [tuple
                      of floats] (GET only).

                    As with standard XSPEC's "tclout covar", this only
                      returns the diagonal and below-diagonal matrix
                      elements.

   criticalDelta -- Critical delta for fit statistic convergence [float].
                      The absolute change in the fit statistic between
                      iterations, less than which the fit is deemed to
                      have converged.

   delta         -- Set fit delta values to be proportional to the
                      parameter value [float].

                 Get: Returns the current proportional setting, or 0.0 if
                        currently using the fixed fit delta values.

                 Set: Enter the constant factor which will multiply the
                        parameter value to produce a fit delta.  A constant
                        factor of 0.0 or negative will turn off the use of
                        proportional fit deltas.

   dof           -- The degrees of freedom for the fit [int] (GET only).

   method        -- The fitting algorithm to use [string].

                    Choices are: "leven", "migrad", "minimize", "monte",
                     "simplex".  The default is "leven".

                    When setting the method, additional arguments for
                      <nFitIterations> and <fit critical delta> may also be
                      entered.  Valid formats for entering multiple
                      arguments are:

                      # Single string
                      Fit.method = "migrad  100 .05"
                      # List of strings
                      Fit.method = ["migrad","100",".05"]
                      # List of strings and numbers
                      Fit.method = ["migrad", 100, .05]
   nIterations   -- The maximum number of fit iterations prior to query [int].

   query         -- The fit query setting [string].
                       "yes": Fit will continue through query.
                       "no" : Fit will end at query.
                       "on" : User will be prompted for "y/n" response.

   statistic     -- Fit statistic value from the most recent fit [float]
                       (GET only).

   statMethod    -- The type of fit statistic in use [string].
                       Valid names: "chi" | "cstat" | "lstat" | "pgstat" |
                          "pstat" | "whittle"
                       To set for individual spectra, add a spectrum number
                           (or range) to the string: ie. Fit.statMethod = "cstat 2"

   statTest      -- The type of test statistic in use [string].
                       Valid names: "ad" | "chi" | "cvm" | "ks" | "pchi" |
                          "runs"
                       To set for individual spectra, add a spectrum number
                           (or range) to the string: ie. Fit.statTest = "ad 2"

   testStatistic     -- Test statistic value from the most recent fit [float]
                       (GET only).

   weight        -- Change the weighting function used in the calculation of
                       chi-sq [string].

                       Available functions: "standard", "gehrels",
                         "churazov", "model"
```

8.7.2 Constructor & Destructor Documentation

8.7.2.1 def __init__ (*self*)

8.7.3 Member Function Documentation

8.7.3.1 def error (*self, argString*)

```
Determine confidence intervals of a fit.

   Input: argString is a string with identical syntax to the standard
     interactive XSPEC error command.
     "[[stopat <ntrial> <toler>] [maximum <redchi>]
           [<delta fit statistic>] [<model param range>...]]"

      where:
      <model param range> =::[<modelName>:]<first param> -
                                  <last param>

   See the XSPEC manual for a more detailed description.
   The results of the error command are stored in the "error" attributes
      of the individual Parameter objects.

   Examples:

   # Estimate the 90% confidence ranges for parameters 1-3
       Fit.error("1-3")
   # Repeat but with delta fit statistic = 9.0, equivalent to the
   # 3 sigma range.
       Fit.error("9.0")
   # Estimate for parameter 3 after setting the number of trials to 20.
   # Note that the tolerance field has to be included (or skipped over).
       Fit.error("stop 20,,3")
```

8.7.3.2 def ftest (*self, chisq2, dof2, chisq1, dof1*)

```
Calculate the F-statistic and its probability given new and old
      values of chisq and number of degrees of freedom (DOF).

   Input:  chisq2 - float
   dof2   - int
   chisq1 - float
   dof1   - int

   Chisq2 and dof2 should come from a new fit, in which an extra model
      component was added to (or a frozen parameter thawed from) the
      model which gave chisq1 and dof1.  If the F-test probability is
      low then it is reasonable to add the extra model component.
      WARNING: it is not correct to use the F-test statistic to test
      for the presence of a line (see Protassov et al 2002, ApJ 571,
      545).

    Returns: The F-test probability [float].
```

8.7.3.3 def goodness (*self, nRealizations =* `100`, *sim =* `False`)

```
Perform a Monte Carlo calculation of the goodness-of-fit.

   nRealizations -- Number of spectra to simulate [int].
   sim           -- If False (default), all simulations are drawn from
           the best fit model parameter values.  If True,
           parameters will be drawn from a Gaussian centered
           on the best fit.
```

8.7.3.4 def improve (*self*)

```
Try to find a new minimum.

   When Fit.method is set to one of the MINUIT algorithms, this
   will run the MINUIT 'improve' command.  This does nothing when
   Fit.method is set to Levenberg-Marquardt.
```

8.7.3.5 def perform (*self*)

```
Perform fit.
```

8.7.3.6 def renorm (*self*, *setting* = None)

```
Renormalize the model to minimize statistic with current parameters

setting -- If None, this will perform an explicit immediate
     renormalization.  Other options determine when
     renormalization will be performed automatically.  They
     are the following strings:

     "auto"   - Renormalize after a model command or parameter
                change, and at the beginning of a fit.

     "prefit" - Renormalize only at the beginning of a fit.

     "none"   - Perform no automatic renormalizations.
```

8.7.3.7 def show (*self*)

```
Show fit information.
```

8.7.3.8 def steppar (*self*, *argString*)

```
Perform a steppar run.

   Generate the statistic "surface" for 1 or more parameters.

   Input: argString is a string with identical syntax to the standard
     interactive XSPEC steppar command.
     "<step spec> [<step spec> ...]" where:

      <step spec> ::= [<log|nolog>] [<current|best>]
    [<modName>:]<param index> <low value> <high value> <# steps>

      See the XSPEC manual for a more detailed description of specs.

    Examples:

      # Step parameter 3 from 1.5 to 2.5 in 10 linear steps
  Fit.steppar("3 1.5 2.5 10")
      # Repeat the above but with logarithmic steps
  Fit.steppar("log")
      # Step parameter 2 linearly from -.2 to .2 in steps of .02
  Fit.steppar("nolog 2 -.2 .2 20")
```

8.7.3.9 def stepparResults (*self*, *arg*)

```
Retrieve values from the most recent steppar run.

   Input: arg  -- argument should either be ’statistic’, ’delstat’,
             or a parameter specifier.  A parameter specifier
             should be a string of the form:
             ’[<modName>:]<parNum>’ or simply an integer <parNum>.

   Returns the requested values as a list of floats.
```

8.7.4 Member Data Documentation

8.7.4.1 bayes

Turn Bayesian inference on or off [string].

```
Valid settings are "on", "off" (default), or "cons".
"cons" turns Bayesian inference on AND gives ALL
parameters a constant prior.  Priors can be set for
parameters individually through the Parameter object’s
’prior’ attribute.
```

8.7.4.2 covariance

The covariance matrix from the most recent fit [tuple of floats] (GET only).

```
As with standard XSPEC’s "tclout covar", this only
  returns the diagonal and below-diagonal matrix
  elements.
```

8.7.4.3 criticalDelta

Critical delta for fit statistic convergence [float].

```
The absolute change in the fit statistic between
iterations, less than which the fit is deemed to
have converged.
```

8.7.4.4 delta

Set fit delta values to be proportional to the parameter value [float].

```
Get: Returns the current proportional setting, or 0.0 if
       currently using the fixed fit delta values.
```

```
Set: Enter the constant factor which will multiply the
       parameter value to produce a fit delta.  A constant
       factor of 0.0 or negative will turn off the use of
       proportional fit deltas.
```

8.7.4.5 dof

The degrees of freedom for the fit [int] (GET only).

8.7.4.6 method

The fitting algorithm to use [string].

```
Choices are: "leven", "migrad", "minimize", "monte",
 "simplex".  The default is "leven".

When setting the method, additional arguments for
  <nFitIterations> and <fit critical delta> may also be
  entered.  Valid formats for entering multiple
  arguments are:

  # Single string
  Fit.method = "migrad  100 .05"
  # List of strings
  Fit.method = ["migrad","100",".05"]
  # List of strings and numbers
  Fit.method = ["migrad", 100, .05]
```

8.7.4.7 nIterations

The maximum number of fit iterations prior to query [int].

8.7.4.8 query

The fit query setting [string].

```
"yes": Fit will continue through query.
"no" : Fit will end at query.
"on" : User will be prompted for "y/n" response.
```

8.7.4.9 statistic

Fit statistic value from the most recent fit [float] (GET only).

8.7.4.10 statMethod

The type of fit statistic in use [string].

```
Valid names: "chi" | "cstat" | "lstat" | "pgstat" |
   "pstat" | "whittle"
To set for individual spectra, add a spectrum number
    (or range) to the string: ie. Fit.statMethod = "cstat 2"
```

8.7.4.11 statTest

The type of test statistic in use [string].

```
Valid names: "ad" | "chi" | "cvm" | "ks" | "pchi" |
   "runs"
To set for individual spectra, add a spectrum number
    (or range) to the string: ie. Fit.statTest = "ad 2"
```

8.7.4.12 testStatistic

Test statistic value from the most recent fit [float] (GET only).

8.7.4.13 weight

Change the weighting function used in the calculation of chi-sq [string].

```
Available functions: "standard", "gehrels",
  "churazov", "model"
```

The documentation for this class was generated from the following file:

- fit.py

8.8 Model Class Reference

Public Member Functions

- def __init__
- def __setattr__
- def __call__
- def energies
- def folded
- def setPars
- def show
- def showList
- def untie
- def values

Public Attributes

- name

  The model name, optional in Xspec.
- nParameters

  Number of parameters in Model object [int].
- expression

  The model expression string, using full component names.
- componentNames

  List of component name strings.
- flux

  A tuple containing the results of the most recent flux calculation for this model.
- lumin

  Same as flux but for luminosity calculations.
- startParIndex

  Global index of the first parameter in this Model object [int].

8.8.1 Detailed Description

```
Xspec model class.

Public instance attributes.  Unless stated otherwise, each is get only.
flux and lumin are implemented as properties.

   expression     -- The model expression string, using full component
                        names.

   name           -- The model name, optional in Xspec.
                        This is an empty string for un-named models.

   <components>   -- Model includes an attribute of type Component for every
                        Xspec component in the model.  The attribute name is
                        the same as the full name of the Xspec component
                        (ie. m=Model("po") produces an m.powerlaw
                        attribute).

   componentNames -- List of component name strings.

   flux           -- A tuple containing the results of the most recent flux
                        calculation for this model.

                     The tuple values are: (value, errLow, errHigh (in
                        ergs/cm^2), value, errLow, errHigh (in photons)).
                        This will be filled in after an AllModels.calcFlux()
                        call ONLY when no spectra are loaded.  Otherwise
                        results are stored in the Spectrum objects.

   lumin          -- Same as flux but for luminosity calculations.
   nParameters    -- Number of parameters in Model object [int].

   startParIndex  -- Global index of the first parameter in this Model
                          object [int].
```

8.8.2 Constructor & Destructor Documentation

8.8.2.1 def __init__ (*self*, *exprString*, *modName* = "", *sourceNum* = 1, *setPars* = None)

```
Model constructor.

New model is automatically added to the AllModels container, with
one Model object constructed (internally) for each data group to
which the model applies.  This function returns the Model object
corresponding to the lowest numbered data group.

exprString -- The model expression string, components may be
         abbreviated.

modName    -- Optional name assigned to model.  Any whitespace in
         string will be removed.  This is required if
         souce number is > 1.

sourceNum  -- Optional integer for model's source number.


setPars    -- Optional initial values for the model's parameters.
         These may be sent in a tuple, list, or dictionary
         (or as a single float or string if only setting the
         first parameter).  Examples:

       # Create a model with all default parameter settings:
       m1 = Model("gauss")

       # Create wabs*powerlaw and initialize pars 1 and 3 to
       #   something other than their default values.
       m2 = Model("wa*po", setPars={1:5.5, 3:".18,,.01,.02"})

       # Create another model named 'b', and reset par 2 to 5.0:
       m3 = Model("wa*bbody", "b", setPars={2:5.0})

       If any mistakes are made with the optional setPars
       parameter arguments, the model will be created using
       all default values.

       You can always reset the parameters later with the
       Model.setPars() method, or directly through the Parameter
       object's 'values' attribute.
```

8.8.3 Member Function Documentation

8.8.3.1 def __call__ (*self*, *parIdx*)

```
Get a Parameter object from the Model.

parIdx -- The parameter index number.  Regardless of the data group
    to which the Model object belongs, its parameters are
    numbered from 1 to nParameters.

Returns the specified Parameter object.
```

8.8.3.2 def __setattr__ (*self*, *attrName*, *value*)

8.8.3.3 def energies (*self*, *spectrumIndex*)

```
Get the Model object's energies array for a given spectrum.

spectrumIndex - The spectrum index number.  If this is 0, it will
          return the energies array used by the default dummy
          response.

Returns a list of energy array elements, the size will be 1 larger than
the corresponding flux array.

This will return the energies array as specified by the
AllModels.setEnergies function if that has been used to override
the response energies array.
```

8.8.3.4 def folded (*self*, *spectrumIndex*)

```
Get the Model object's folded flux array for a given spectrum.

spectrumIndex -- The spectrum index number.  This number should
           be 0 if model is not presently applied to any
           spectra (ie. in the "off" state).

Returns a list of folded flux array elements.
```

8.8.3.5 def setPars (*self*, *parVals*)

```
Change the value of multiple parameters in a single function call.

This is a quick way to change multiple parameter values at a time
since only a SINGLE model recalculation will be performed at the end.
In contrast, when parameter values are changed through the individual
parameter objects, the model is recalculated after EACH parameter
change.  (See also AllModels.setPars(), for changing multiple parameters
belonging to multiple model objects.)

parVals    -- An arbitrary number of parameter values.  These may
        be listed singly (as floats or strings), or collected
        into tuple, list or dictionary containers.
        Dictionaries must be used if parameters are not in
        consecutive order, in which case the parameter index
        number is the dictionary key.

Examples:  Assume we have a model object m1 with 5 parameters.

   Simplest case: change only the parameter values (and not the
     auxiliary values, 'sigma', 'min', 'bot', etc.), and change
     them in consecutive order.

     # Pass in 1 or more floats
     m1.setPars(5.5, 7.83, 4.1e2)  # changes pars 1-3
     m1.setPars(2.0, 1.3e-5, -.05, 6.34, 9.2)  # changes all 5 pars

   Still changing only the parameter values, but skipping over some.

     m1.setPars(.02, 4.4, {5:3.2e5})  # changes pars 1-2, 5
     m1.setPars({2:3.0, 4:-1.2})  # changes pars 2, 4
     m1.setPars({2:1.8}, 9.3, 5.32)  # changes pars 2, 3, 4

   Now also change the auxiliary values for some of the parameters.
     Pass in a STRING containing "<val>,<sigma>,<min>,<bottom>,<top>,
     <max>"  This uses the same syntax as Standard XSPEC's "newpar"
     command.  Aux values can be skipped by using multiple commas.

     # This sets a new <val>, <sigma>, and <max> for parameter 1, and
     # a new <val> of 5.3 for parameter 2.
     m1.setPars(".3,.01,,,,100", 5.3)

     # This sets all new auxiliary values for parameter 3.
     m1.setPars({3:".8 -.01 1e-4 1e-3 1e5 1e6"})
```

8.8.3.6 def show (*self*)

```
Display information for a single Model object.
```

8.8.3.7 def showList ( )

```
Show the list of all available XSPEC model components.
```

8.8.3.8 def untie (*self*)

```
Remove links for all parameters in Model object
```

8.8.3.9 def values (*self, spectrumIndex*)

```
Get the Model object's values array for a given spectrum.

spectrumIndex -- The spectrum index number.  This number should
           be 0 if model is not presently applied to any
```

```
          spectra (ie. in the "off" state).

Returns the values array as a list.
```

8.8.4 Member Data Documentation

8.8.4.1 componentNames

List of component name strings.

8.8.4.2 expression

The model expression string, using full component names.

8.8.4.3 flux

A tuple containing the results of the most recent flux calculation for this model.

```
 The tuple values are: (value, errLow, errHigh (in
    ergs/cm^2), value, errLow, errHigh (in photons)).
    This will be filled in after an AllModels.calcFlux()
    call ONLY when no spectra are loaded.  Otherwise
    results are stored in the Spectrum objects.
```

8.8.4.4 lumin

Same as flux but for luminosity calculations.

8.8.4.5 name

The model name, optional in Xspec.

```
 This is an empty string for un-named models.
```

8.8.4.6 nParameters

Number of parameters in Model object [int].

8.8.4.7 startParIndex

Global index of the first parameter in this Model object [int].

The documentation for this class was generated from the following file:

- model.py

8.9 ModelManager Class Reference

Public Member Functions

- def __init__
- def __call__
- def __iadd__
- def __isub__
- def addPyMod
- def calcFlux
- def calcLumin
- def clear
- def eqwidth
- def setEnergies
- def initpackage
- def lmod
- def setPars
- def show
- def simpars

Public Attributes

- sources

  A dictionary containing the currently active <source number>="">:<model name>=""> assignments.
- systematic

  The fractional model systematic error.

8.9.1 Detailed Description

```
Models container.

This is a singleton - only 1 instance allowed

Public attributes:

   sources          -- A dictionary containing the currently active
                          <source number>:<model name> assignments.
                          If the model has no name, <model name> will
                          be an empty string.  (GET only)

   systematic       -- The fractional model systematic error.
                          This will be added in quadrature to the error
                          on the data when evaluating chi-squared.  The
                          default value is zero.
```

8.9.2 Constructor & Destructor Documentation

8.9.2.1 def __init__ (*self*)

8.9.3 Member Function Documentation

8.9.3.1 def __call__ (*self*, *groupNum*, *modName* = "")

```
Get Model objects from the AllModels container.

groupNum -- The data group number to which the Model object corresponds.

modName  -- Optional string containing the Model's name (if any).

Returns the Model object.
```

8.9.3.2 def __iadd__ (*self*, *modelInfo*)

```
Define a new model and add it to the AllModels container.

This operation is equivalent to the Model class constructor,
except that it does not return a Model object.

modelInfo -- A string containing the model expression (component
        names may be abbreviated).   The model will be
        unnamed and assigned to source number = 1.

     OR

     If supplying a model name and a source number, this
     should be a tuple with:

        modelInfo[0] = model expression string
        modelInfo[1] = model name string
        modelInfo[2] = source number
```

8.9.3.3 def __isub__ (*self*, *modName*)

```
Remove all copies of the given model from the AllModels container.

modName -- The name of the model to be removed, or an empty string if
      the model has no name.   If set to "*", this will behave
      like the clear() function and remove all models.
```

8.9.3.4 def addPyMod (*self*, *func*, *parInfo*, *compType*, *calcsErrors* = `False`, *spectrumDependent* = `False`)

```
Add a user-defined Python model function to XSPEC's models library.
```

```
   This provides a way to add local models written in Python to XSPEC.  It
   performs the same role as the combination of 'initpackage'/'lmod' commands
   do for C/C++/Fortran local models.  The first 3 arguments (func, parInfo,
   and compType) are mandatory.

   func        -- The user-defined model function (Python type = 'function').
            Function must define at least 3 arguments for energies,
            parameters, and flux.

            A optional fourth argument may be added if your model
            calculates flux errors, and a fifth if your model
            requires that XSPEC pass it the spectrum number.

   parInfo     -- A tuple of strings.  One string for each parameter your
            model requires.  The format of these strings is identical
            to what is placed in a 'model.dat' file (see Appendix C of
             the XSPEC manual).

   compType    -- A string telling XSPEC the type of your model.
            Currently allowed types:  'add', 'mul', 'con'

   calcsErrors  -- OPTIONAL.  If your model function also calculates model errors,
             set this to True.

   spectrumDependent  -- OPTIONAL.  Set this to TRUE only if your model function
                      has an explicit dependence on the spectrum.

   Example usage:  A local additive model written in Python, named 'myModel',
             which takes parameters 'par1' and 'par2':

     def myModel(engs, pars, flux):
         [... model code, fill in
           flux array based on input
           engs and pars arrays ...]

     myModelParInfo=("par1  \"\" 2.0  -10.0  -9.0  9.0  10.0  0.01",
                     "par2  keV  1e-3 1e-5  1e-5  100. 200. .01" )

     AllModels.addPyMod(myModel, myModelParInfo, 'add')
```

8.9.3.5 def calcFlux (*self*, *cmdStr*)

```
Calculate the model flux for a given energy range.

cmdStr -- A string containing the energy limit values and
  optional error specifiers.  This follows the same
  syntax rules as the standard XSPEC flux command.

The flux will be calculated for all loaded spectra, and the results
will be stored in the Spectrum objects' flux attribute.  If no
spectra are loaded, the flux will be stored in the Model objects'
flux attribute.
```

8.9.3.6 def calcLumin (*self, cmdStr*)

```
Calculate the model luminosity for a given energy range and redshift.

cmdStr -- A string containing the energy limit values and
    optional error specifiers.  This follows the same
    syntax rules as the standard XSPEC lumin command.

The lumin will be calculated for all loaded spectra, and the results
will be stored in the Spectrum objects' lumin attribute.  If no
spectra are loaded, the flux will be stored in the Model objects'
lumin attribute.
```

8.9.3.7 def clear (*self*)

```
Remove all models.
```

8.9.3.8 def eqwidth (*self, component, rangeFrac =* `None`, *err =* `False`, *number =* `None`, *level =* `None`)

```
Calculate the equivalent width of a model component.

Please see the Standard XSPEC Manual for a discussion on how the eqwidth
of a component is calculated.

component -- An integer specifying the model component number for which
       to calculate the eqwidth (left-most component is 1).  If
       the component belongs to a NAMED model, then this must be
       a STRING of the form "<modelName>:<compNumber>".

rangeFrac -- Determines the energy range for the continuum calculation:
       from E(1-<rangeFrac>) to E(1+<rangeFrac>) where E is
       the location of the peak of the photon spectrum.  The
       initial default rangeFrac is 0.05.  Setting this will
       change the future default value.

err       -- If set to True, errors will be estimated on the equivalent
       width calculation.  This will also require the setting of
       the "number" and "level" arguments.

number    -- Only set this if "err" = True.  This determines the number
       of sets of randomized parameter values to draw to make
       the error estimation. [int]

level     -- Only set this if "err" = True.  The error algorithm will
        order the equivalent widths of the <number> sets of
        parameter values, and the central <level> percent will
        determine the error range.  [float]

The results of the most recent eqwidth calculation are stored as
attributes of the currently loaded Spectrum objects.
```

8.9.3.9 def initpackage (*self*, *packageName*, *modDescrFile*, *dirPath* = `None`, *udmget* = `False`)

```
Initialize a package of local models.

Use this method to compile your local model source code and build
a library, which can then be loaded into XSPEC with the 'lmod' method.

packageName  -- The name of the model package [string].
          The name should be all lower-case and contain NO
          numerals or spaces.  The local models library file
          will be based upon this name, and this is also the
          name you will use when loading the library with the
          'lmod' method.

modDescrFile -- Name of your local model description file [string].
          This file is typically named 'lmodel.dat', but you're
          free to name it something else.

dirPath      -- Optional directory path to your local models [string].
          This may be an absolute or relative path.  If you
          don't enter this argument, XSPEC will look in the
          directory given by the LOCAL_MODEL_DIRECTORY in your
          Xspec.init start-up file.

udmget       -- Optional flag for when your models need to call XSPEC's
          udmget function [bool].  Udmget is a function for
          allocating dynamic memory in Fortran routines, and
          is no longer used within XSPEC itself.  If this
          flag is set to 'True', initpackage will copy the
          necessary files and build the udmget function within
          your local models directory.
```

8.9.3.10 def lmod (*self*, *packageName*, *dirPath* = `None`)

```
Load a local models library.

packageName -- The name of the model package to be loaded.  This
         is the same name that is the first argument in
         the initpackage command.

dirPath     -- An optional string argument specifying the (absolute or
         relative) path to the local model directory.  If this
         argument is not entered, Xspec will look in the
         directory given by the LOCAL_MODEL_DIRECTORY in the
         Xspec.init start-up file.
```

8.9.3.11 def setEnergies (*self*, *arg1*, *arg2* = `None`)

```
Specify new energy binning for model fluxes.

Supply an energy binning array to be used in model evalutations in place
of the associated response energies, or add an extension to the response
```

```
energies.

arg1 -- A string containing either:
  "<range specifier> [<additional range specifiers>...]"
  "<name of input ascii file>"
  "extend"  [This option also uses arg2]
  "reset"

where the first <range specifier> ::=
        <lowE> <highE> <nBins> log|lin
<additional range specifier> ::= <highE> <nBins> log|lin
This uses the same syntax as standard XSPEC's "energies"
command.  Values can be delimited by spaces or commas.

arg2 -- Only needed when arg1 is "extend", this requires an extension
  specifier string of the form:
  "low|high <energy> <nBins> log|lin"

All energies are in keV.  Multiple ranges may be specified to allow for
varied binning in different segments of the array, but note that no gaps
are allowed in the overall array.  Therefore only the first range
specifier accepts a <lowE> parameter.  Additional ranges will
automatically begin at the <highE> value of the previous range.

With the "extend" option, the specifier string supplied to arg2 will
extend the existing response energy array by an additional <nBins> to
the new <energy>, in either the high or low direction.

Once an energy array is specified, it will apply to all models and will
be used in place of any response energy array (from actual or dummy
responses) for calculating and binning the model flux.  It will also
apply to any models that are created after it is specified.  To turn off
this behavior and return all models back to using their response
energies, set arg1 to "reset".

Arg1 can also be the name of an ascii text file containing a custom
energy array.  To see the proper file format, and for more details in
general about the energies command, please see the standard XSPEC
manual.

Examples:

  # Create an array of 1000 logarithmic-spaced bins, from .1 to 50. keV
  AllModels.setEnergies(".1 50. 1000 log")
  # Change it to 500 bins
  AllModels.setEnergies(",,500")
  # Now restore original response energies, but with an extension of the
  #   high end to 75.0 keV with 100 additional linear bins.
  AllModels.setEnergies("extend","high,75.,100 lin")
  # Return to using original response energies with no extensions.
  AllModels.setEnergies("reset")
```

8.9.3.12 def setPars (*self*, *args*)

```
Change the value of multiple parameters from multiple model objects
```

```
     with a single function call.

This is a quick way to change multiple parameter values at a time
since only a SINGLE recalculation will be performed at the end.
In contrast, when parameter values are changed through the individual
parameter objects, the model is recalculated after EACH parameter
change.  (If all the parameters belong to a single model object,
you can also use the Model.setPars() function.)

args    -- An arbitrary number model objects and parameter values.
     The first argument must be model object, followed by
     one or more of its new parameter values.  Additional
     groups of model objects and parameter values may follow.

   The parameter values follow the same syntax rules as with
     the single Model.setPars() function.  They can be listed
     singly (as floats or strings), or collected into tuple,
     list, or dictionary containers.  Dictionaries must be used
     when parameters are not in consecutive order, in which case
     the parameter index number is the dictionary key.

   Parameter indices are local to each model object.  That is,
     they are always numbered from 1 to N where N is the number
     of parameters in the model object.

Examples:

    Assume we've already assigned a 3 parameter model to 2 data groups:
      m1 = AllModels(1)
      m2 = AllModels(2)

    # Various ways of changing parameters in consecutive order.

    # This changes pars 1-2 in m1 and 1-3 in m2:
    AllModels.setPars(m1, .4, "1.3 -.01", m2, "5.3 ,,3.0e-4", 2.2, 1.9)
    #   ...and these 2 examples do the exact same thing as above:
    valList = [.4, "1.3 -.01"]
    valTuple = ("5.3 ,,3.0e-4", 2.2, 1.9)
    AllModels.setPars(m1, valList, m2, valTuple)
    AllModels.setPars(m1, valList, m2,"5.3 ,,3.0e-4", [2.2, 1.9])

    # Parameters in non-consecutive order, must use Python
    #   dictionaries:

    # Change parameter 2 in m1, parameter 1 and 3 in m2:
    AllModels.setPars(m1, {2:8.3}, m2, {1:0.99, 3:"7.15 -.01"})
    # ...same thing as above:
    AllModels.setPars(m1, {2:8.3}, m2, 0.99, {3:"7.15 -.01"})

    Note that identical syntax is used for model objects belonging
      to different sources.  All of the above examples are still valid
      had we obtained m1 and m2 like this:

      m1 = Model("wabs*pow", "firstMod", 1)
      m2 = Model("gauss", "secondMod", 2)
```

8.9.3.13 def show (*self*, *parIDs =* `None`)

```
Show all or a subset of Xspec model parameters.

parIDs -- An optional string specifying a range of parameters as
    with Xspec's "show parameter" function.  If no string is
    supplied, this will show all parameters in all models.
```

8.9.3.14 def simpars (*self*)

```
Create a list of simulated parameter values.

Values are drawn from a multivariate normal distribution based on the
covariance matrix from the last fit, or from Monte Carlo Markov chains
if they are loaded.  This method is identical to doing 'tclout simpars'
in standard XSPEC.

Returns a tuple of the simulated parameter values.
```

8.9.4 Member Data Documentation

8.9.4.1 sources

A dictionary containing the currently active <source number>="">:<model name>=""> assignments.

```
If the model has no name, <model name> will
be an empty string.  (GET only)
```

8.9.4.2 systematic

The fractional model systematic error.

```
This will be added in quadrature to the error
on the data when evaluating chi-squared.  The
default value is zero.
```

The documentation for this class was generated from the following file:

- model.py

8.10 Parameter Class Reference

Public Member Functions

- def __init__
- def untie
- def __float__
- def __add__
- def __radd__
- def __iadd__
- def __mul__
- def __rmul__
- def __imul__

Public Attributes

- name

  Name of Parameter (GET only).
- values

  List of value floats [val,delta,min,bot,top,max].
- sigma

  The Parameter fit sigma (-1.0 when not applicable) (GET only).
- frozen

  Boolean, if True then parameter is frozen.
- link

  Link expression string (empty if not linked).
- index

  Position of the parameter within the Model object.
- unit

  An optional string for the parameter's units (GET only).
- error

  A tuple containing the results of the most recent fit error command performed on the parameter (GET only).
- prior

  A tuple containing the settings for the prior used when Bayesian inference is turned on.

8.10.1 Detailed Description

```
Model or response parameter class.

Public instance attributes, implemented as properties.

   name   -- Name of Parameter (GET only).
   values -- List of value floats [val,delta,min,bot,top,max].
             This may be set with:
                string:        x.values = "3.2,,,,1e2, 1e3"
                single float:  x.values = 4.1  (sets 'val' only)
                tuple:         x.values = 8.2,.02, -10.
                list:          x.values = [8.2,.02, -10.]
             Note that Tuple and List input do not allow the use
                of consecutive commas for argument spacing.

   sigma  -- The Parameter fit sigma (-1.0 when not applicable) (GET only).

   frozen -- Boolean, if True then parameter is frozen.

   link   -- Link expression string (empty if not linked).

   index  -- Position of the parameter within the Model object.
               (The first parameter has index = 1)  Note that this is the
               same value that would be used to obtain a Parameter object
               from its Model, ie: par = mod(<index>)
               (GET only).

   unit   -- An optional string for the parameter's units (GET only).

   error  -- A tuple containing the results of the most recent fit error
               command performed on the parameter (GET only).
               The tuple values are: (error low bound, error high bound,
                  error status code string)

   prior  -- A tuple containing the settings for the prior used when
               Bayesian inference is turned on.

               Get: Returns a tuple containing:
                 (<priorType>, <optional hyperparameters>)

               Set with:
                  string:   <priorType>
                  or tuple: (<priorType>, <optional hyperparameters>)
                  Valid priorTypes are "cons", "exp", "jeffreys", "gauss".
                  Hyperparameters should be entered as floats.
```

8.10.2 Constructor & Destructor Documentation

8.10.2.1 def __init__ (*self, parName, parStrategy*)

```
Parameter constructor.

Not intended for stand-alone creation.  This should only be called
from within Model, Component, or Response classes.

parName -- Parameter name
```

8.10.3 Member Function Documentation

8.10.3.1 def __add__ (*self*, *other*)

8.10.3.2 def __float__ (*self*)

8.10.3.3 def __iadd__ (*self*, *other*)

8.10.3.4 def __imul__ (*self*, *other*)

8.10.3.5 def __mul__ (*self*, *other*)

8.10.3.6 def __radd__ (*self*, *other*)

8.10.3.7 def __rmul__ (*self*, *other*)

8.10.3.8 def untie (*self*)

```
Remove parameter link (if any)
```

8.10.4 Member Data Documentation

8.10.4.1 error

A tuple containing the results of the most recent fit error command performed on the parameter (GET only).

```
 The tuple values are: (error low bound, error high bound,
    error status code string)
```

8.10.4.2 frozen

Boolean, if True then parameter is frozen.

8.10.4.3 index

Position of the parameter within the Model object.

```
 (The first parameter has index = 1)  Note that this is the
 same value that would be used to obtain a Parameter object
 from its Model, ie: par = mod(<index>)
 (GET only).
```

8.10.4.4 link

Link expression string (empty if not linked).

8.10.4.5 name

Name of Parameter (GET only).

8.10.4.6 prior

A tuple containing the settings for the prior used when Bayesian inference is turned on.

```
Get: Returns a tuple containing:
  (<priorType>, <optional hyperparameters>)

Set with:
   string:   <priorType>
   or tuple: (<priorType>, <optional hyperparameters>)
   Valid priorTypes are "cons", "exp", "jeffreys", "gauss".
   Hyperparameters should be entered as floats.
```

8.10.4.7 sigma

The Parameter fit sigma (-1.0 when not applicable) (GET only).

8.10.4.8 unit

An optional string for the parameter's units (GET only).

8.10.4.9 values

List of value floats [val,delta,min,bot,top,max].

```
This may be set with:
   string:        x.values = "3.2,,,,1e2, 1e3"
   single float:  x.values = 4.1  (sets 'val' only)
   tuple:         x.values = 8.2,.02, -10.
   list:          x.values = [8.2,.02, -10.]
Note that Tuple and List input do not allow the use
   of consecutive commas for argument spacing.
```

The documentation for this class was generated from the following file:

- parameter.py

8.11 PlotManager Class Reference

Public Member Functions

- def __init__

- def __call__
- def addCommand
- def delCommand
- def iplot
- def noID
- def setGroup
- def setID
- def setRebin
- def show
- def x
- def xErr
- def y
- def yErr
- def model
- def backgroundVals

Public Attributes

- add

  Turn on/off the display of individual additive components [bool].
- area

  Toggle displaying the data divided by the response effective area for each channel [bool].
- background

  Toggle displaying the background spectrum (if any) when plotting data [bool].
- commands

  Custom plot commands to be appended to Xspec-generated commands.
- device

  The plotting device name [string].
- perHz

  Toggle displaying Y-axis units per Hz when using wavelength units for X-axis [bool].
- redshift

  Apply a redshift to the X-axis energy or wavelength values [float].
- splashPage

  When set to False, the usual XSPEC version and build data information will not be printed to the screen when the first plot window is initially opened [bool].
- xAxis

  X-Axis Units [string].
- xLog

  Set the x-axis to logarithmic or linear for energy or wavelength plots [bool].
- yLog

  See xLog.

8.11.1 Detailed Description

```
Xspec plotting class.

This is a singleton - only 1 instance allowed

Public instance attributes:

   add        -- Turn on/off the display of individual additive
                    components [bool].

   area       -- Toggle displaying the data divided by the response
                    effective area for each channel [bool].

   background -- Toggle displaying the background spectrum (if any)
                    when plotting data [bool].

   commands   -- Custom plot commands to be appended to Xspec-generated
                    commands.

                 Get: Returns a tuple of the currently entered command
                      strings.
                 Set: Replaces all commands with the new tuple of
                      strings.

                 To remove ALL plot commands, set to an empty tuple, ie:
                    Plot.commands = ()

                 For inserting and deleting individual commands, use
                    addCommand and delCommand functions.

   device     -- The plotting device name [string].

   perHz      -- Toggle displaying Y-axis units per Hz when using
                    wavelength units for X-axis [bool].

   redshift   -- Apply a redshift to the X-axis energy or wavelength
                    values [float].
                 This will multiply X-axis energies by a factor of (1+z)
                    to allow for viewing in the source frame.  Y-axis values
                    will be equally affected in plots which are normalized
                    by energy or wavelength.  Note that this is not
                    connected in any way to redshift parameters in the model
                    (or the setplot id redshift parameter) and should only
                    be used for illustrative purposes.

   splashPage -- When set to False, the usual XSPEC version and build data
                    information will not be printed to the screen when the
                    first plot window is initially opened [bool].

   xAxis      -- X-Axis Units [string].
                    Valid options are:   "channel"
                      (energies)         "keV", "MeV", "GeV", "Hz"
                      (wavelengths)      "angstrom", "cm", "micron", nm"

                    These are case-insensitive and may be abbreviated.
                    This setting also affects the ignore/notice range
                    interpretation.

   xLog       -- Set the x-axis to logarithmic or linear for energy or
                    wavelength plots [bool].

                 xLog has no effect on plots in channel space.  xLog
                    and yLog will not work for model-related plots
                    (eg. model, ufspec, and their variants) as their axes
                    are always set to log scale.

   yLog       -- See xLog.
```

8.11.2 Constructor & Destructor Documentation

8.11.2.1 def __init__ (*self, deviceStr*)

8.11.3 Member Function Documentation

8.11.3.1 def __call__ (*self, panes*)

```
Display the plot.

Input 1 or more plot command strings.

Examples:

   Single Plots:
      Plot("data")
      Plot("model")
      Plot("ufspec")

   Multiple Plots (or single plots taking additional arguments):
      Plot("data","model","resid")
      Plot("data model resid")
      Plot("data,model,resid")
      Plot("data","model m1")  # Plots data and a model named "m1".

To repeat a plot using the previously entered arguments,
   simply do: Plot()
```

8.11.3.2 def addCommand (*self, cmd*)

```
Add a plot command [string] to the end of the plot commands list.
```

8.11.3.3 def backgroundVals (*self, plotGroup =* `1`*, plotWindow =* `1`)

```
Return a list of background data values for a plot group and plot window

Background value arrays only exist for data plots when the
```

```
   Plot.background flag is set to True.
```

8.11.3.4 def delCommand (*self*, *num*)

```
Remove a plot command by (1-based) number [int].

This is intended for removal of single commands.  To remove ALL
commands, set the Plot.commands attribute to an empty tuple, ie:
   Plot.commands = ()
```

8.11.3.5 def iplot (*self*, *panes*)

```
Display the plot and leave it in interactive plotting mode.

   This function takes the same arguments and syntax as when
      displaying plots in the regular mode (through Plot's
      __call__ method).  Examples:

      Plot.iplot("data")   # 1-panel data plot
      Plot.iplot("data model")    # 2-panel data and model
      Plot.iplot()                # Repeats the previous plot.
```

8.11.3.6 def model (*self*, *plotGroup* = `1`, *plotWindow* = `1`)

```
Return a list of Y-coordinate model values for a plot group and plot window
```

8.11.3.7 def noID (*self*)

```
Turn off the plotting of line IDs.
```

8.11.3.8 def setGroup (*self*, *groupStr*)

```
Define a range of spectra to be in the same plot group.

Input argument is a string specifying one or more ranges, delimited
   by commas and/or spaces.

   Examples:
      "1-3  4-6" : Spectra 1-3 in plot group 1, 4-6 in group 2.
      "1,2 4"    : Spectra 1, 2, and 4 are each now in their own group.
      "1-**"     : All spectra are in a single plot group.

      None       : If input argument is Python's None variable, all
           plot grouping will be removed.
```

8.11.3.9 def setID (*self*, *temperature* = None, *emissivity* = None, *redshift* = None)

```
Switch on plotting of line IDs.

All input arguments are floats and are optional.  If they are omitted
   they will retain their previous values.

   temperature  -- Selects the temperature of the APEC line list.
   emissitivity -- Only lines with emissivities above this setting will
              be displayed.
   redshift     -- Line display will be redshifted by this amount.

To turn off plotting of line IDs, use the noID() function.
```

8.11.3.10 def setRebin (*self*, *minSig* = None, *maxBins* = None, *groupNum* = None, *errType* = None)

```
Define characteristics used in rebinning the data (for plotting
   purposes ONLY).

All input arguments are optional.  If they are omitted they will retain
   their previous values.

   minSig   -- Bins will be combined until this minimum significance
          is reached (in units of sigma).  [float]
   maxBins  -- The maximum number of bins to combine in attempt to
          reach minSig.  [int]
   groupNum -- The plot group number to which this setting applies.
          If number is negative, it will apply to ALL plot
          groups.  [int]
   errType  -- Specifies how to calculate the error bars on the new
          bins.  Valid entries are "quad", "sqrt", "poiss-1",
          "poiss-2", "poiss-3".  [string]  See the "setplot"
          description in the XSPEC manual for more information.
```

8.11.3.11 def show (*self*)

```
Display current plot settings
```

8.11.3.12 def x (*self*, *plotGroup* = 1, *plotWindow* = 1)

```
Return a list of X-coordinate data values for a plot group and plot window
```

8.11.3.13 def xErr (*self*, *plotGroup* = 1, *plotWindow* = 1)

```
Return a list of X-coordinate errors for a plot group and plot window
```

8.11.3.14 def y (*self*, *plotGroup* = 1, *plotWindow* = 1)

```
Return a list of Y-coordinate data values for a plot group and plot window
```

8.11.3.15 def yErr (*self*, *plotGroup* = `1`, *plotWindow* = `1`)

```
Return a list of Y-coordinate errors for a plot group and plot window
```

8.11.4 Member Data Documentation

8.11.4.1 add

Turn on/off the display of individual additive components [bool].

8.11.4.2 area

Toggle displaying the data divided by the response effective area for each channel [bool].

8.11.4.3 background

Toggle displaying the background spectrum (if any) when plotting data [bool].

8.11.4.4 commands

Custom plot commands to be appended to Xspec-generated commands.

```
 Get: Returns a tuple of the currently entered command
      strings.
 Set: Replaces all commands with the new tuple of
      strings.

 To remove ALL plot commands, set to an empty tuple, ie:
    Plot.commands = ()

 For inserting and deleting individual commands, use
    addCommand and delCommand functions.
```

8.11.4.5 device

The plotting device name [string].

8.11.4.6 perHz

Toggle displaying Y-axis units per Hz when using wavelength units for X-axis [bool].

8.11.4.7 redshift

Apply a redshift to the X-axis energy or wavelength values [float].

```
 This will multiply X-axis energies by a factor of (1+z)
```

```
    to allow for viewing in the source frame.  Y-axis values
    will be equally affected in plots which are normalized
    by energy or wavelength.  Note that this is not
    connected in any way to redshift parameters in the model
    (or the setplot id redshift parameter) and should only
    be used for illustrative purposes.
```

8.11.4.8 splashPage

When set to False, the usual XSPEC version and build data information will not be printed to the screen when the first plot window is initially opened [bool].

8.11.4.9 xAxis

X-Axis Units [string].

```
Valid options are:   "channel"
  (energies)         "keV", "MeV", "GeV", "Hz"
  (wavelengths)      "angstrom", "cm", "micron", nm"
These are case-insensitive and may be abbreviated.
This setting also affects the ignore/notice range
interpretation.
```

8.11.4.10 xLog

Set the x-axis to logarithmic or linear for energy or wavelength plots [bool].

```
xLog has no effect on plots in channel space.  xLog
   and yLog will not work for model-related plots
   (eg. model, ufspec, and their variants) as their axes
   are always set to log scale.
```

8.11.4.11 yLog

See xLog.

The documentation for this class was generated from the following file:

- plot.py

8.12 Response Class Reference

Public Member Functions

- def __init__

- def setPars
- def show

Public Attributes

- gain

  A response model object (class RModel) for applying a shift in response file gain.
- arf

  Get/Set the arf filename string.
- chanEnergies

  Tuple of floats, the detector channel energies in keV.
- energies

  Tuple of floats, the photon energies in keV which are stored in the MATRIX extension.
- rmf

  The response file name string.
- sourceNumber

  The 1-based source number for which the response is assigned.

8.12.1 Detailed Description

```
Detector response class.

Public instance attributes implemented as properties, these are GET only
   unless specified otherwise.

   arf          -- Get/Set the arf filename string.
                      Enter None or empty string to remove an existing arf.

   chanEnergies -- Tuple of floats, the detector channel energies in keV.
                      These are the energies normally stored in the
                      EBOUNDS extension.

   energies     -- Tuple of floats, the photon energies in keV which are
                      stored in the MATRIX extension.

   gain         -- A response model object (class RModel) for applying
                      a shift in response file gain.

                      (Also see Response.setPars() for setting multiple
                      gain parameters at a time.)

                      When gain is turned ON, it creates two variable
                      fit Parameter object members:
                         gain.slope    (default = 1.0)
                         gain.offset   (default = 0.0)

                      To turn gain ON simply assign a value to EITHER
                      parameter, ie.:
                         gain.slope = 1.05

                      This automatically also creates a gain.offset
                      parameter with default value 0.0, which you may
                      want to re-adjust. Examples:
                         gain.offset = .02
                         gain.offset.values = ".015,.001,,,,0.1"

                      'slope' and 'offset' are of the same type as regular
                      model parameters, and therefore have the same
                      functions, attributes, and syntax rules for setting
                      values.  (See the Parameter class help for more
                      details.)

                      To turn gain OFF, call its off() method:
                         gain.off()

                      gain.off() restores the response to its original state,
                      and renders the 'slope' and 'offset' parameters
                      inaccessible.

   rmf          -- The response file name string.

   sourceNumber -- The 1-based source number for which the response is
                      assigned.
                   This is normally always 1 unless multiple sources are
                      loaded for multiple-model evaluation.
```

8.12.2 Constructor & Destructor Documentation

8.12.2.1 def __init__ (*self*, *parent*, *respTuple*)

```
Construct a Response object.

Intended for creation by a Spectrum object only.
```

8.12.3 Member Function Documentation

8.12.3.1 def setPars (*self*, *seqPars*)

```
Set multiple response parameters with a single function call.

Similar to the Model.setPars() function, this allows multiple
response parameters to be changed with just a SINGLE recalculation
performed at the end.

seqPars    -- An arbitrary number of CONSECUTIVE parameter values
      to be matched 1-to-1 with the response model's
      parameters.

Currently just 1 response model ('gain') is available, which has
```

```
2 response parameters ('slope' and 'offset').

Examples:

   s = Spectrum("file1")
   resp = s.response

   # 'gain' is off by default and response parameters don't yet exist.
   #   The following call automatically turns 'gain' on and creates
   #   both 'slope' and 'offset' parameters even though it is only
   #   assigning to 'slope'.  'offset' will retain its default value
   #   of 0.0.
   resp.setPars(1.05)  # Equivalent to doing: resp.gain.slope = 1.05

   # This is equivalent to: resp.gain.slope = .995
   #                        resp.gain.offset = .08
   #   except that the recalculation is only performed at the end
   #   rather than after each parameter is changed:
   resp.setPars(.995, .08)

   # Can also assign auxiliary values by passing 1 or 2 string
   #   arguments.
   resp.setPars("1.1,,.02,.02,1.8,1.8","-.05,,-2,-2")

   # Remove gain and restore response to original state:
   resp.gain.off()
```

8.12.3.2 def show (*self*)

```
Display response information including (optional) response parameters.
```

8.12.4 Member Data Documentation

8.12.4.1 arf

Get/Set the arf filename string.

```
 Enter None or empty string to remove an existing arf.
```

8.12.4.2 chanEnergies

Tuple of floats, the detector channel energies in keV.

```
 These are the energies normally stored in the
 EBOUNDS extension.
```

8.12.4.3 energies

Tuple of floats, the photon energies in keV which are stored in the MATRIX extension.

8.12.4.4 gain

A response model object (class RModel) for applying a shift in response file gain.

```
 (Also see Response.setPars() for setting multiple
gain parameters at a time.)

When gain is turned ON, it creates two variable
fit Parameter object members:
   gain.slope   (default = 1.0)
   gain.offset  (default = 0.0)

To turn gain ON simply assign a value to EITHER
parameter, ie.:
   gain.slope = 1.05

This automatically also creates a gain.offset
parameter with default value 0.0, which you may
want to re-adjust. Examples:
   gain.offset = .02
   gain.offset.values = ".015,.001,,,,0.1"

'slope' and 'offset' are of the same type as regular
model parameters, and therefore have the same
functions, attributes, and syntax rules for setting
values.  (See the Parameter class help for more
details.)

To turn gain OFF, call its off() method:
   gain.off()

gain.off() restores the response to its original state,
and renders the 'slope' and 'offset' parameters
inaccessible.
```

8.12.4.5 rmf

The response file name string.

8.12.4.6 sourceNumber

The 1-based source number for which the response is assigned.

```
This is normally always 1 unless multiple sources are
   loaded for multiple-model evaluation.
```

The documentation for this class was generated from the following file:

- response.py

8.13 RModel Class Reference

Public Member Functions

- def __init__
- def __getattribute__
- def __setattr__
- def off

Public Attributes

- parameterNames

  List of the response model's parameter names (get only).
- isOn

  Boolean flag showing the On/Off status of the RModel (get only).

8.13.1 Detailed Description

```
Response Model class.

Response models are functions which act upon the detector RMF.   XSPEC
currently has just one response model: 'gain', which is a built-in attribute
of the Response class.  RModel objects are not intended for stand-alone
creation: its __init__ function should be considered private.

Public instance attributes.

   <parameters>   -- When RModel is ON, it contains an attribute of type
                     Parameter for every parameter in the model.  An
                     RModel is turned ON by a 'set' operation on ANY
                     of its parameters.  For example with the 'gain'
                     RModel:

                     resp.gain.offset = .03

                     automatically creates 'offset' AND 'slope'
                     parameters if they don't already exist ('slope'
                     would be initialized to its default value of 1.0).
                     The shift is then applied immediately to the
                     Response object 'resp'.

                     When RModel is OFF (see the RModel.off() method),
                     the parameters are not accessible.

   isOn           -- Boolean flag showing the On/Off status of the RModel
                     (get only).
   parameterNames -- List of the response model's parameter names
                        (get only).
```

8.13.2 Constructor & Destructor Documentation

8.13.2.1 def __init__ (*self, resp, parNames, rmodName*)

```
RModel constructor.

   Intended for internal use only.
```

8.13.3 Member Function Documentation

8.13.3.1 def __getattribute__ (*self, name*)

8.13.3.2 def __setattr__ (*self, attrName, value*)

8.13.3.3 def off (*self*)

```
Remove response parameters and turn the model OFF.

The Response is restored to its original state.
```

8.13.4 Member Data Documentation

8.13.4.1 isOn

Boolean flag showing the On/Off status of the RModel (get only).

8.13.4.2 parameterNames

List of the response model's parameter names (get only).

The documentation for this class was generated from the following file:

- response.py

8.14 Spectrum Class Reference

Public Member Functions

- def __init__
- def dummyrsp

- def ignore
- def ignoredString
- def notice
- def noticedString
- def show

Public Attributes

- areaScale

  The Spectrum area scaling factor.
- background

  Get/Set the spectrum's background.
- backScale

  The Spectrum background scaling factor.
- cornorm

  Get/Set the normalization of a spectrum's correction file [float].
- correction

  Get/Set the correction file.
- dataGroup

  The data group to which the spectrum belongs [int].
- energies

  Tuple of pairs of floats (also implemented as tuples) giving the E_Min and E_Max of each noticed channel.
- eqwidth

  Tuple of 3 floats containing the results of the most recent eqwidth calculation for this spectrum (performed with the AllModels.eqwidth method).
- exposure

  The exposure time keyword value [float].
- fileName

  The spectrum's file name [string].
- flux

  A tuple containing the results of the most recent flux calculation for this spectrum.
- ignored

  A list of the currently ignored (1-based) channel numbers.
- index

  The spectrum's current index number within the AllData container [int].
- isPoisson

  Boolean flag, true if spectrum has Poisson errors.
- lumin

  Similar to flux, the results of the most recent luminosity calculation.

- multiresponse

  Get/Set detector response ARRAY elements when using multiple sources.
- noticed

  A list of the currently noticed (1-based) channel numbers.
- rate

  A tuple containing the total Spectrum rates in counts/sec.
- response

  Get/Set the detector response.
- values

  Tuple of floats containing the spectrum rates for noticed channels in counts/cm^2-sec.
- variance

  Tuple of floats containing the variance of each noticed channel.

8.14.1 Detailed Description

```
Spectral data class.

Public instance attributes (implemented as properties).  Unless stated
otherwise, each is GET only.

   areaScale -- The Spectrum area scaling factor.
                Either a single float (if file stores it as a keyword),
                or a Tuple of floats (if file stores column).

   background  -- Get/Set the spectrum's background.

                  Get: Returns the Background object associated with the
                         Spectrum.  If Spectrum has no background object,
                         this will raise an Exception.

                  Set: Supply a background filename [string].
                         This will become the new background to the Spectrum
                         object, and any previously existing background will
                         be removed.  If string is empty, all whitespace,
                         or the Python None variable, the background (if
                         any) will be removed.

   backScale -- The Spectrum background scaling factor.
                Either a single float (if file stores it as a keyword),
                or a Tuple of floats (if file stores column).

   cornorm    -- Get/Set the normalization of a spectrum's correction file
                 [float].

   correction -- Get/Set the correction file.

                  Get: Returns the Spectrum's current correction information
                         as an object of class Background.  This raises an
                         Exception if Spectrum has no correction.

                  Set: Enter the filename string for the new correction.
```

```
                         This will remove any previously existing
                         correction.  Returns the new correction info
                         as an object of class Background.
                         If string is "none", empty, or all whitespace,
                         the current correction will be removed and this
                         will return None.

dataGroup -- The data group to which the spectrum belongs [int].

energies  -- Tuple of pairs of floats (also implemented as tuples)
                giving the E_Min and E_Max of each noticed channel.

eqwidth   -- Tuple of 3 floats containing the results of the most recent
                eqwidth calculation for this spectrum (performed with the
                AllModels.eqwidth method).

             The results are stored as:
                  [0] - eqwidth calculation
                  [1] - eqwidth error lower bound
                  [2] - eqwidth error upper bound
                The error bounds will be 0.0 if no error calculation was
                performed, and all will be 0.0 if eqwidth wasn't
                performed for this spectrum.

exposure  -- The exposure time keyword value [float].

fileName  -- The spectrum's file name [string].

flux      -- A tuple containing the results of the most recent flux
                calculation for this spectrum.

             The tuple values are:
                (value, errLow, errHigh (in ergs/cm^2), value, errLow,
                errHigh (in photons)) for each model applied to the
                spectrum.

ignored   -- A list of the currently ignored (1-based) channel numbers.

index     -- The spectrum's current index number within the AllData
                container [int].

isPoisson -- Boolean flag, true if spectrum has Poisson errors.

lumin     -- Similar to flux, the results of the most recent luminosity
                calculation.

multiresponse -- Get/Set detector response ARRAY elements when
                   using multiple sources.

                This is for use only when assigning multiple responses
                   to a spectrum, for multi-source/multi-model analysis.
                   For standard single-source analysis, use the
                   "response" attribute instead.

                You must provide an array index for all multiresponse
                   get/set operations. Note that array indices ARE 0-BASED,
                   so multiresponse[0] corresponds to source 1. Examples:
                   # Get the response assigned to source 1.
                   # This particular call is the same as doing
                   # "r1 = s.response"
                   r1 = spec.multiresponse[0]

                   # Get the response for the second source.
                   # Can only do this with multiresponse.
                   r2 = spec.multiresponse[1]

                   # Define a third source by adding a new response:
                   spec.multiresponse[2] = "myResp3.pha"

                   # Now remove the response for the second source:
                   spec.multiresponse[1] = None


noticed   -- A list of the currently noticed (1-based) channel numbers.

rate      -- A tuple containing the total Spectrum rates in counts/sec.

             The tuple consists of:
                [0] - current net rate (w/ background subtracted),
                [1] - net rate variance,
                [2] - total rate (without background),
                [3] - predicted model rate

response  -- Get/Set the detector response.

                Use this for standard SINGLE-SOURCE analysis.
                To add other responses for multi-source and multi-model
                analysis, use the "multiresponse" attribute.

                Get: Returns a Response object, or raises an
                       Exception if none exists

                Set: Supply a response filename string.  To remove
                        a response, supply an empty string or None.


values    -- Tuple of floats containing the spectrum rates for noticed
                channels in counts/cm^2-sec.

variance  -- Tuple of floats containing the variance of each noticed
                channel.
```

8.14.2 Constructor & Destructor Documentation

8.14.2.1 def __init__ (*self*, *dataFile*)

```
Construct a Spectrum object.

Read in a spectrum and any associated background, response and
arf files.  Spectrum is automatically added to the AllData container.
```

```
dataFile - Spectral data filename [string].
```

8.14.3 Member Function Documentation

8.14.3.1 def dummyrsp (*self*, *lowE* = None, *highE* = None, *nBins* = None, *scaleType* = None, *chanOffset* = None, *chanWidth* = None, *sourceNum* = 1)

```
Create a dummy response for this spectrum only.

   Input arguments, all are optional:

     lowE  - Input response energy lower bound, in keV. [float]
     highE - Input response energy higher bound, in keV. [float]
     nBins - Number of bins into which the energy range is divided
        [int].
     scaleType  - "log" or "lin" [string]
     chanOffset - Starting value of dummy channel energies. [float]
     chanWidth  - Energy width of the channel bins.  [float]
             If this is set to 0, the dummy response
             can only be used for evaluating model arrays,
             and not for fitting to spectra.
     sourceNum  - Optional source number for the dummy response. [int]

   Examples:

# All values are optional, use keywords to enter values
# non-consecutively.  Unspecified values revert to the
# current defaults.
s = Spectrum("dataFile.pha")
s.dummyrsp(.3, 30., 100, chanWidth=.5)
s.dummyrsp(highE = 50., sourceNum = 2)
s.dummyrsp(.1,10.,100,"lin",.0, 1.0, 1)

Initial defaults:  lowE = .1, highE = 50., nBins = 50, scaleType = "log"
   chanOffset = .0, chanWidth = .0, sourceNum = 1
The defaults for lowE, highE, nBins, scaleType, and chanOffset will be
modified for each explicit new entry.  chanWidth always defaults to 0
and sourceNum always defaults to 1.

To remove the spectrum's dummy response(s) and restore actual
responses (if any), call AllData.removeDummyrsp().
```

8.14.3.2 def ignore (*self*, *ignoreRange*)

```
Ignore a range of the spectrum by channels or energy/wavelengths.

ignoreRange -- String specifying the channel range to ignore.
         This follows the same syntax as used in the standard
         Xspec "ignore" command. If the numbers are floats
         rather than ints, they will be treated as energies or
         wavelengths (depending on the Plot settings).
         Note that "bad" will not work from here, as it can
         only be applied to ALL of the loaded spectra.

         To apply range(s) to multiple spectra, use the AllData
         ignore function.
```

8.14.3.3 def ignoredString (*self*)

```
Return a string of ignored channel ranges.

This produces a string in compact (hyphenated) form, which can be
used as input to a subsequent 'ignore' command.  Example:

   If ignored channels are [1,3,4,5,7],
     this will output "1 3-5 7".
```

8.14.3.4 def notice (*self*, *noticeRange*)

```
Notice a range of the spectrum by channels or energy/wavelengths.

noticeRange -- String specifying the channel range to notice.
         This follows the same syntax as used in the standard
         Xspec "notice" command. If the numbers are floats
         rather than ints, they will be treated as energies or
         wavelengths (depending on the Plot settings).  If the
         string is "all", it will notice all channels in
         spectrum.

         To apply range(s) to multiple spectra, use the AllData
         notice function.
```

8.14.3.5 def noticedString (*self*)

```
Return a string of noticed channel ranges.

This produces a string in compact (hyphenated) form, which can be
used as input to a subsequent 'notice' command.  Example:

   If noticed channels are [1,3,4,5,7],
     this will output "1 3-5 7".
```

8.14.3.6 def show (*self*)

```
Display information for this Spectrum object
```

8.14.4 Member Data Documentation

8.14.4.1 areaScale

The Spectrum area scaling factor.

```
Either a single float (if file stores it as a keyword),
or a Tuple of floats (if file stores column).
```

8.14.4.2 background

Get/Set the spectrum's background.

```
Get: Returns the Background object associated with the
       Spectrum.  If Spectrum has no background object,
       this will raise an Exception.

Set: Supply a background filename [string].
       This will become the new background to the Spectrum
       object, and any previously existing background will
       be removed.  If string is empty, all whitespace,
       or the Python None variable, the background (if
       any) will be removed.
```

8.14.4.3 backScale

The Spectrum background scaling factor.

```
Either a single float (if file stores it as a keyword),
or a Tuple of floats (if file stores column).
```

8.14.4.4 cornorm

Get/Set the normalization of a spectrum's correction file [float].

8.14.4.5 correction

Get/Set the correction file.

```
Get: Returns the Spectrum's current correction information
       as an object of class Background.  This raises an
       Exception if Spectrum has no correction.
Set: Enter the filename string for the new correction.
       This will remove any previously existing
       correction.  Returns the new correction info
```

```
        as an object of class Background.
        If string is "none", empty, or all whitespace,
        the current correction will be removed and this
        will return None.
```

8.14.4.6 dataGroup

The data group to which the spectrum belongs [int].

8.14.4.7 energies

Tuple of pairs of floats (also implemented as tuples) giving the E_Min and E_Max of each noticed channel.

8.14.4.8 eqwidth

Tuple of 3 floats containing the results of the most recent eqwidth calculation for this spectrum (performed with the AllModels.eqwidth method).

```
 The results are stored as:
      [0] - eqwidth calculation
      [1] - eqwidth error lower bound
      [2] - eqwidth error upper bound
    The error bounds will be 0.0 if no error calculation was
    performed, and all will be 0.0 if eqwidth wasn't
    performed for this spectrum.
```

8.14.4.9 exposure

The exposure time keyword value [float].

8.14.4.10 fileName

The spectrum's file name [string].

8.14.4.11 flux

A tuple containing the results of the most recent flux calculation for this spectrum.

```
 The tuple values are:
    (value, errLow, errHigh (in ergs/cm^2), value, errLow,
    errHigh (in photons)) for each model applied to the
    spectrum.
```

8.14.4.12 ignored

A list of the currently ignored (1-based) channel numbers.

8.14.4.13 index

The spectrum's current index number within the AllData container [int].

8.14.4.14 isPoisson

Boolean flag, true if spectrum has Poisson errors.

8.14.4.15 lumin

Similar to flux, the results of the most recent luminosity calculation.

8.14.4.16 multiresponse

Get/Set detector response ARRAY elements when using multiple sources.

```
This is for use only when assigning multiple responses
   to a spectrum, for multi-source/multi-model analysis.
   For standard single-source analysis, use the
   "response" attribute instead.
You must provide an array index for all multiresponse
   get/set operations. Note that array indices ARE 0-BASED,
   so multiresponse[0] corresponds to source 1. Examples:
   # Get the response assigned to source 1.
   # This particular call is the same as doing
   # "r1 = s.response"
   r1 = spec.multiresponse[0]
   # Get the response for the second source.
   # Can only do this with multiresponse.
   r2 = spec.multiresponse[1]
   # Define a third source by adding a new response:
   spec.multiresponse[2] = "myResp3.pha"
   # Now remove the response for the second source:
   spec.multiresponse[1] = None
```

8.14.4.17 noticed

A list of the currently noticed (1-based) channel numbers.

8.14.4.18 rate

A tuple containing the total Spectrum rates in counts/sec.

```
The tuple consists of:
   [0] - current net rate (w/ background subtracted),
   [1] - net rate variance,
   [2] - total rate (without background),
   [3] - predicted model rate
```

8.14.4.19 response

Get/Set the detector response.

```
Use this for standard SINGLE-SOURCE analysis.
To add other responses for multi-source and multi-model
analysis, use the "multiresponse" attribute.

Get: Returns a Response object, or raises an
       Exception if none exists

Set: Supply a response filename string.  To remove
        a response, supply an empty string or None.
```

8.14.4.20 values

Tuple of floats containing the spectrum rates for noticed channels in counts/cm^2-sec.

8.14.4.21 variance

Tuple of floats containing the variance of each noticed channel.

The documentation for this class was generated from the following file:

- spectrum.py

8.15 XspecSettings Class Reference

Public Member Functions

- def __init__
- def addModelString
- def delModelString
- def closeLog
- def openLog
- def show

Public Attributes

- abund

  Get/Set the abundance table used in the plasma emission and photoelectric absorption models [string].
- allowNewAttributes

  Get/Set the flag which allows the setting of new instance attributes for ALL PyXspec classes [bool].

- chatter

  Get/Set the console chatter level [int].
- logChatter

  Get/Set the log chatter level [int].
- cosmo

  Get/Set the cosmology values.
- log

  Get only: Returns the currently opened log file object, or None if no log file is open (also see the openLog and closeLog methods).
- modelStrings

  XSPEC's internal database of <string_name>, <string_value> pairs for settings which may be accessed by model functions.
- parallel

  An attribute for controlling the number of parallel processes in use during various XSPEC contexts.
- seed

  Re-seed and re-initialize XSPEC's random-number generator with the supplied integer value (SET only).
- version

  The version strings for PyXspec and standard XSPEC.
- xsect

  Change the photoelectric absorption cross-sections in use [string].

8.15.1 Detailed Description

```
Storage class for Xspec settings.

This is a singleton - only 1 instance allowed

Public instance attributes (implemented as properties):

   abund      -- Get/Set the abundance table used in the plasma emission and
                   photoelectric absorption models [string].

                   Valid tables: angr, aspl, feld, aneb, grsa, wilm, lodd,
                                   file <filename>

   allowNewAttributes -- Get/Set the flag which allows the setting of new
                  instance attributes for ALL PyXspec classes [bool].

                  This is False by default, and is intended to catch the
                     user's attention if they misspell an attribute name
                     when attempting to set it.  Under normal Python
                     behavior, a misspelling would simply create a new
                     attribute and issue no warnings or errors.

                  You must make sure this flag is set to True if you
```

```
                     genuinely wish to add new attributes.

   chatter    -- Get/Set the console chatter level [int].
   logChatter -- Get/Set the log chatter level [int].

   cosmo      -- Get/Set the cosmology values.

           Get: Returns a tuple of floats containing (H0, q0, l0), where
                  H0 is the Hubble constant in km/(s-Mpc),
                  q0 is the deceleration parameter, and
                  l0 is the cosmological constant.

           Set: Enter a single string containing one or more of
                    H0, q0, l0.  Examples:

                  Xset.cosmo = "100"  # sets H0 to 100.0
                  Xset.cosmo = ",0"   # sets q0 to 0.0
                  Xset.cosmo = ",,0.7"  # sets l0 to 0.7
                  Xset.cosmo = "50 .5 0." # sets H0=50.0, q0=0.5, l0=0.0

   log        -- Get only: Returns the currently opened log file object,
                    or None if no log file is open (also see the openLog
                    and closeLog methods).

   modelStrings -- XSPEC's internal database of <string_name>,
                      <string_value> pairs for settings which may be
                      accessed by model functions.

                      Get: Returns a tuple of tuples, the inner tuples
                           being composed of <string_name>,<string_value>
                           string pairs.

                      Set: Replaces ENTIRE database with user-supplied
                           new database.  Input may be a dictionary of
                           <string_name>:<string_value> entries, or a tuple
                           of (<string_name>,<string_value>) tuples.

                      For inserting and deleting INDIVIDUAL string
                         name and value pairs, use the addModelString and
                         delModelString methods.

   parallel   -- An attribute for controlling the number of parallel
                    processes in use during various XSPEC contexts.
                    Examples:

                  # Use up to 4 parallel processes during
                  #   Levenberg-Marquardt fitting.
                  Xset.parallel.leven = 4

                  # Use up to 4 parallel processes during
                  #   Fit.error() command runs.
                  Xset.parallel.error = 4

                  # Reset all contexts to single process usage.
                  Xset.parallel.reset()

   seed       -- Re-seed and re-initialize XSPEC's random-number generator
```

```
                   with the supplied integer value (SET only).

   version    -- The version strings for PyXspec and standard XSPEC.
                   GET only, this returns a tuple containing:
                   [0] - The PyXspec version string
                   [1] - Standard XSPEC's version string

   xsect      -- Change the photoelectric absorption cross-sections in use
                   [string].

                   Available options: "bcmc", "obcm", "vern"
```

8.15.2 Constructor & Destructor Documentation

8.15.2.1 def __init__ (*self*)

8.15.3 Member Function Documentation

8.15.3.1 def addModelString (*self, key, value*)

```
Add a key,value pair of strings to XSPEC's internal database.

   This database provides a way to pass string values to certain
   model functions which are hardcoded to search for "key".
   (See the XSPEC manual description for the "xset" command for a
   table showing model/key usage.)

   If the key,value pair already exists, it will be replaced with
   the new entries.
```

8.15.3.2 def closeLog (*self*)

```
Close Xspec's current log file.
```

8.15.3.3 def delModelString (*self, key*)

```
Remove a key,value pair from XSPEC's internal string database.
```

8.15.3.4 def openLog (*self, fileName*)

```
Open a file and set it to be Xspec's log file.

   fileName -- The name of the log file.

   If Xspec already has an open log file, it will close it.
   Returns a Python file object for the new log file.
   Once opened, the log file object is also stored as the
   Xset.log attribute.
```

8.15.3.5 def show (*self*)

8.15.4 Member Data Documentation

8.15.4.1 abund

Get/Set the abundance table used in the plasma emission and photoelectric absorption models [string].

```
 Valid tables: angr, aspl, feld, aneb, grsa, wilm, lodd,
                file <filename>
```

8.15.4.2 allowNewAttributes

Get/Set the flag which allows the setting of new instance attributes for ALL PyXspec classes [bool].

```
 This is False by default, and is intended to catch the
    user's attention if they misspell an attribute name
    when attempting to set it.  Under normal Python
    behavior, a misspelling would simply create a new
    attribute and issue no warnings or errors.

 You must make sure this flag is set to True if you
    genuinely wish to add new attributes.
```

8.15.4.3 chatter

Get/Set the console chatter level [int].

8.15.4.4 cosmo

Get/Set the cosmology values.

```
 Get: Returns a tuple of floats containing (H0, q0, l0), where
        H0 is the Hubble constant in km/(s-Mpc),
        q0 is the deceleration parameter, and
        l0 is the cosmological constant.

 Set: Enter a single string containing one or more of
          H0, q0, l0.  Examples:

       Xset.cosmo = "100"  # sets H0 to 100.0
```

```
Xset.cosmo = ",0"   # sets q0 to 0.0
Xset.cosmo = ",,0.7"  # sets l0 to 0.7
Xset.cosmo = "50 .5 0." # sets H0=50.0, q0=0.5, l0=0.0
```

8.15.4.5 log

Get only: Returns the currently opened log file object, or None if no log file is open (also see the openLog and closeLog methods).

8.15.4.6 logChatter

Get/Set the log chatter level [int].

8.15.4.7 modelStrings

XSPEC's internal database of <string_name>, <string_value> pairs for settings which may be accessed by model functions.

```
Get: Returns a tuple of tuples, the inner tuples
     being composed of <string_name>,<string_value>
     string pairs.

Set: Replaces ENTIRE database with user-supplied
     new database.  Input may be a dictionary of
     <string_name>:<string_value> entries, or a tuple
     of (<string_name>,<string_value>) tuples.

For inserting and deleting INDIVIDUAL string
   name and value pairs, use the addModelString and
   delModelString methods.
```

8.15.4.8 parallel

An attribute for controlling the number of parallel processes in use during various XSPEC contexts.

```
Examples:
se up to 4 parallel processes during
 Levenberg-Marquardt fitting.
t.parallel.leven = 4
se up to 4 parallel processes during
 Fit.error() command runs.
t.parallel.error = 4
eset all contexts to single process usage.
t.parallel.reset()
```

8.15.4.9 seed

Re-seed and re-initialize XSPEC's random-number generator with the supplied integer value (SET only).

8.15.4.10 version

The version strings for PyXspec and standard XSPEC.

```
GET only, this returns a tuple containing:
[0] - The PyXspec version string
[1] - Standard XSPEC's version string
```

8.15.4.11 xsect

Change the photoelectric absorption cross-sections in use [string].

```
Available options: "bcmc", "obcm", "vern"
```

The documentation for this class was generated from the following file:

- xset.py

Index